Combustion and Flame 159 (2012) 2388–2397

Contents lists available at SciVerse ScienceDirect

Combustion and Flame

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

Accelerating multi-dimensional combustion simulations using GPU and hybrid explicit/implicit ODE integration

Yu Shi [a,*], William H. Green [a], Hsi-Wu Wong [b], Oluwayemisi O. Oluwole [b,*]

[a] Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[b] Aerodyne Research Inc., Billerica, MA 01821, USA

ARTICLE INFO

Article history:
Received 20 September 2011
Received in revised form 1 December 2011
Accepted 14 February 2012
Available online 8 March 2012

Keywords:
Combustion modeling
GPU
Parallel computation
CUDA
Chemical kinetics

ABSTRACT

Simulating multi-dimensional combustion with detailed kinetics often requires solving a large number of ordinary differential equation (ODE) problems at each global time step. In many cases, the ODE integrations account for the bulk of the total wall-clock time for the simulation. This paper introduces CHEMEQ2-GPU – a new explicit stiff ODE solver (based on the existing CHEMEQ2 solver) that exploits the parallel architecture of the modern graphics processing unit (GPU) to accelerate ODE integration in multi-dimensional combustion simulations. We also demonstrate efficient application of the CPU and GPU as co-processors, for further speedup. We describe a hybrid explicit/implicit ODE solver approach that combines the strengths of both solver types running simultaneously on the GPU and CPU, respectively. A dynamic load balancing scheme was used to assign the kinetics ODE integrations over all grid points to either the CPU-based implicit solver DVODE (which is the more efficient solver for highly stiff grid points) or CHEMEQ2-GPU (more efficient for moderately stiff or non-stiff grid points). We demonstrate CHEMEQ2-GPU and the hybrid approach in 3-D simulations of homogeneous charge compression ignition (HCCI) engines. The test cases applied two different n-heptane reaction mechanisms (a large detailed model and a small skeletal model) and three different mesh sizes. Engine simulations were performed using KIVA-CHEMKIN. CHEMEQ2 was about 2–3 times faster than DVODE, with similar prediction accuracy. The CHEMEQ2-GPU speedup relative to CHEMEQ2 increased linearly with the number of grid points for the range of meshes tested in this work. Assuming ideal linear scaling of simulation time with number of processors, the speed of CHEMEQ2-GPU on the Tesla C2050 GPU was equivalent to CHEMEQ2 running on approximately 13 parallel 2.8 GHz CPU processors for the finest mesh; and the hybrid solver approach was equivalent to CHEMEQ2 on ~15 such CPU processors. In summary, CHEMEQ2-GPU provided the additional computing power of 14 parallel CPU processors (for the finest mesh tested) and the hybrid solver approach demonstrated a method to efficiently apply these additional co-processors with existing CPU cores for combustion simulations. CHEMEQ2-GPU scales favorably with the number of grid points and is available by request to the authors. This work presents opportunities for further development, particularly in CPU/GPU load balancing algorithms.

1. Introduction

Combustion simulations are known to be time-intensive, largely due to the associated finite-rate chemical kinetics computations, which typically account for over 90% of the total simulation time. Several approaches have been developed for accelerating these kinetics computations, generally by model size reduction and/or distributed/high performance computing (see recent review by Lu and Law [1]). The latter approach has been particularly revolutionary in enabling the performance of large-scale multi-dimensional combustion simulations that were previously considered intractable, using massively parallel central processing unit (CPU) clusters/architectures. The basic concept of distributed computing is to first identify portions of the simulation algorithm that involve large amounts of expensive, but independent computations. One is then able to reduce the total simulation time by performing these independent computations in parallel – each portion on a different processor, versus a single-processor simulation where all of these computations must be performed serially. Although some overhead costs are incurred due to necessary inter-processor communications, distributed computing often accelerates the parallelized component of the simulation by a factor that is nearly equal to the number of processors used. Indeed, "massively parallel computing"

* Corresponding authors. Present address: Hwys 60 & 123, PLB-210A, Bartlesville, OK 74004, USA (Y. Shi).
E-mail addresses: yushi92@gmail.com, yu.shi@conocophillips.com (Y. Shi), oluwoleo@aerodyne.com (O.O. Oluwole).

0010-2180/$ - see front matter © 2012 The Combustion Institute. Published by Elsevier Inc. All rights reserved.
doi:10.1016/j.combustflame.2012.02.016

(i.e., distributed computing using hundreds to thousands of parallel processors) has been applied to enable large-scale, three-dimensional direct numerical simulation (DNS) of turbulent combustion with detailed chemical kinetics [2]. However, massively parallel simulations remain the exception rather than the norm in the combustion modeling community. Most parallel combustion simulations tend to be limited to no more than a few tens of processors; and for most engineers, particularly in small to medium-sized companies or research organizations, this number is much smaller. This is largely due to the high monetary costs and system administration required for such massively parallel CPU clusters. On the other hand, the graphics processing unit (GPU) has shown rapidly increasing capacity for massively parallel computing with much less system maintenance and much lower monetary cost, thanks to the fast-growing video game industry. The modern GPU is not only a graphics engine but also a highly parallel programmable processor with higher peak arithmetic and memory bandwidth than its CPU counterpart [3]. Further, advances in general purpose GPU computing (such as the invention of NVIDIA CUDA [4]) have enabled programming of complex scientific problems on GPU cores and decreased the learning curve for porting software codes designed for CPUs to GPUs.

Perhaps the most beneficial perspective for scientific computing is that addition of the GPU extends the computing power of existing CPU resources. So the GPU should not be viewed as an alternative, but rather as a supplement for the CPU. The work presented here is based on this perspective.

1.1. GPU Computing in combustion modeling

Although GPUs offer the potential for significantly improved computational capabilities, combustion simulation methods and algorithms have generally not been updated to exploit this emerging technology. Algorithm components that are to be parallelized must be reprogrammed to be executed on the GPU, which may give the impression of a steep learning/implementation curve and serve as a deterrent to potential users. We recently demonstrated methods to exploit the GPU for combustion simulations by applying relatively minor modifications to existing CPU-based simulation algorithms [5,6]. Those methods enable the inclusion of very large kinetic mechanisms ($\gg$100 species) in combustion simulations, since the kinetics computation time grows at a much slower rate with increasing mechanism size (when using the GPU-based modifications versus the standard CPU-only methods). The present paper addresses the goal of applying GPU computing to accelerate large-scale multi-dimensional combustion simulations, where smaller kinetic mechanisms ($\leqslant$100 species, approximately) may be used and the large number of computational grid points is of primary concern. These simulations are most often performed using semi-implicit (with operator-splitting) methods for solving the system of governing partial differential equations (PDEs), due to the prohibitively high memory requirements of fully implicit methods. Fully explicit methods are also applied in some cases, when the stiffness introduced by the chemical kinetics mechanism can be efficiently reduced [1]. Spafford et al. [7] recently described a GPU-based approach for accelerating the kinetics computations in such simulations. Therefore, this paper focuses on multi-dimensional combustion simulation using semi-implicit methods, where kinetics ODE integrations must be performed at a large number of grid points at each global (transport) time step in the simulation. We present a newly developed explicit stiff ODE solver that exploits the parallel computing capabilities of the modern GPU to accelerate these expensive computations. The new GPU-based solver (called CHEMEQ2-GPU) applies the methods implemented in CHEMEQ2 [8], with the modifications described later in this paper, and is available by request to the authors. We also describe a hybrid explicit/implicit ODE solver approach that can be applied to further accelerate multi-dimensional combustion simulations, using the GPU and CPU as parallel co-processors.

The rest of the paper is organized as follows: in Section 2, we describe CHEMEQ2-GPU and the hybrid ODE solver approach; then in Section 3 we demonstrate the performance of CHEMEQ2-GPU and the hybrid solver approach for 3-D simulations of n-heptane combustion in an HCCI engine; finally, in Section 4 we summarize the work and suggest directions for future work.

2. Approach

2.1. Basics of GPU computing

The details of GPU computing are beyond the scope of this work, but a very brief working summary is provided in this subsection. The subject is treated in greater detail in Refs. [9–12].

The basic unit of program execution on a GPU is a "thread" and modern GPUs consist of several "cores", each capable of generating several threads for parallel computing. Computations to be performed on the GPU are programmed as "kernels", which are conceptually similar to subroutines or functions in the Fortran or C programming languages. Execution of a kernel function is performed in parallel as multiple threads that each map to a different piece of data. For instance, to simultaneously compute $f(x)$ for several values of x, a kernel function would be written (in a GPU programming language, such as CUDA or OpenCL) that accepts as input all values of x at which $f(x)$ is desired and parallelizes the multiple evaluations of $f(x)$ across all available threads. Each thread will perform the same set of operations (instructions) to compute $f(x)$, but each for a different value of x – the so-called "single instruction multiple thread" (SIMT) program paradigm. Additionally, all other functions or subroutines called during the execution of the $f(x)$ function must be programmed in a GPU programming language as "device" functions executed on the GPU.

It is also useful to note that GPU memory is separate from CPU memory and data required for execution of kernel functions must be communicated from the CPU to the GPU (and vice versa to use the GPU results on the CPU). On the GPU, threads are organized into "blocks" with *shared* memory (fast access, but limited capacity); and threads in different blocks can communicate with one another via the *global* memory (higher capacity, slower access due to latency), but cannot access shared memory across blocks.

Finally, GPUs come with different performance specifications that should be considered when selecting a GPU for a particular scientific computing application. GPUs with more cores generally offer greater parallelism (since they offer more threads) and many GPUs now feature several hundreds of cores. Also, double-precision arithmetic capability is particularly important for combustion simulations.

2.2. Semi-implicit combustion simulation

In semi-implicit reacting flow PDE integration, operator splitting decouples the numerical integrations of the stiff chemical kinetic equations and the (non-stiff) transport equations. A stiff solver is then used to integrate the kinetics ODEs (Eq. (1)) and a non-stiff, computationally cheaper solver can be used to solve the fluid dynamics equations [13,14].

$$\frac{\partial \phi^j}{\partial t} = \omega(\phi^j) + Const_j \tag{1}$$

$\phi^j \equiv$ set of variables describing the chemical state at jth point in discretized spatial domain; typically ϕ^j = (temperature, pressure, species mass fractions) at point j; ω = model for chemical reactions

Note that the kinetics aspect of the simulation consists entirely of solving Eq. (1) at each grid point, as a function of the local reaction conditions. Eq. (1) is often stiff and computing its solution can account for greater than 90% of the total simulation time. Therefore, it is natural to parallelize the kinetics computations so that each available GPU thread performs the ODE integrations for a subset of the grid points in the computational domain. This means that a kernel function must be written that can accept as input the full set of required ODE integrator inputs for all of the grid points and parallelize the integrations across available threads. Sufficient memory must also be allocated on the GPU to perform the integrations at all of the grid points.

Eq. (1) is most often solved using implicit ODE integration methods, since they are the most efficient class of methods for the numerical integration of highly stiff ODEs such as those often encountered in combustion kinetics. However, advanced implicit ODE solvers, such as DVODE [15], apply sophisticated algorithms to determine proper time step sizes and to control integration errors. The complex logical flow of the control algorithm in implicit ODE solvers results in many divergent instructions on processors during execution time, which does not affect CPU performance (due to large on-chip cache) but can significantly degrade performance in GPU computing (with much smaller cache size). Therefore, implicit integration is more efficiently performed on the CPU (or GPU for a single grid point as demonstrated in our previous work [5]) and porting existing CPU-based implicit ODE solvers to the GPU through parallelization over grid points remains a challenging task that may not yield much benefit.

On the other hand, explicit integration methods are often more efficient than implicit methods for ODE systems that are only moderately stiff or non-stiff. Additionally, explicit methods typically involve simple algorithm logical flows and are well-suited to GPU implementation. The tradeoff is that the efficiency of explicit methods degrades as stiffness increases – smaller time steps are required to retain accuracy and avoid instability; and explicit methods ultimately become impractical for extremely stiff systems.

2.3. Exploiting GPU and hybrid explicit/implicit ODE integration

2.3.1. CHEMEQ2-GPU: a new GPU-based explicit stiff ODE solver

The CHEMEQ2 solver developed by Mott et al. employs an α-QSS method, which is A-stable for linear problems and second-order accurate [8]. It is a single-point and single-step stiff ODE integrator that requires only the information at the current time step to integrate and advance to the next time step. Compared to the multi-point methods normally found in many implicit stiff ODE solvers, such as DVODE, this unique feature of the explicit CHEMEQ2 solver results in minimal start-up costs when applied in multi-dimensional combustion simulations. We briefly describe the α-QSS method and the CHEMEQ2 algorithm, which is implemented in CHEMEQ2-GPU with a few modifications that are noted in the following discussion.

Consider a general form of the species equation in chemical kinetics integration:

$$\frac{dy_i}{dt} = q_i - p_i y_i \tag{2}$$

where q and py_i represent the construction and destruction rates of species i, respectively, and y_i is the species mass fraction. If q and p are constant, solving Eq. (2) yields an exact solution:

$$y_i(t) = y_i^0 e^{-pt} + \frac{q}{p}(1 - e^{-pt}) \tag{3}$$

If q and p are slowly varying when integrating from 0 to Δt, Eq. (3) provides a good approximation for $y_i(\Delta t)$ using q and p values at $t = 0$. Evaluating Eq. (3) at $t = \Delta t$ and reformulating the expression yields:

$$\begin{aligned} y_i(\Delta t) &= y_i^0 + \frac{\Delta t(q_i - p_i y_i^0)}{1 + \alpha_i p_i \Delta t}, \quad \text{where } \alpha_i \\ &= \frac{1 - (1 - e^{-p_i \Delta t})/(p_i \Delta t)}{1 - e^{-p_i \Delta t}} \end{aligned} \tag{4}$$

A predictor–corrector method is used to solve Eq. (4):

$$\begin{aligned} y^P &= y^0 + \frac{\Delta t(q^0 - p^0 y^0)}{1 + \alpha^0 p^0 \Delta t}, \text{ predictor (exponential Euler)} \\ y^C &= y^0 + \frac{\Delta t(q^* - p^* y^0)}{1 + \alpha^* p^* \Delta t}, \text{ corrector} \end{aligned} \tag{5}$$

where α is a function of $p\Delta t$ as in Eq. (4). where superscript 0 represents the initial values. Superscripts P and C indicate predicted and corrected terms. The "starred" terms can be either evaluated using the predicted values or a combination of the predicted and initial terms. The formula of the "starred" terms distinguishes different QSS methods. The α-QSS method in CHEMEQ2 takes the form:

$$\begin{aligned} q^* &= \alpha q^P + (1 - \alpha) q^0 \\ p^* &= \frac{1}{2}(p^P + p^0) \end{aligned} \tag{6}$$

In an internal integration step, a solution is accepted if the difference of the predicted y_i value and the corrected y_i value is within a user-specified error tolerance ε; the original CHEMEQ2 uses

$$\|y_i^C - y_i^P\| \leqslant \varepsilon y_i^C \tag{7}$$

If Eq. (7) is not satisfied, the time step size is reduced until the convergence condition is met. The initial time step size is determined as the fastest kinetic timescale among all the species i, estimated from:

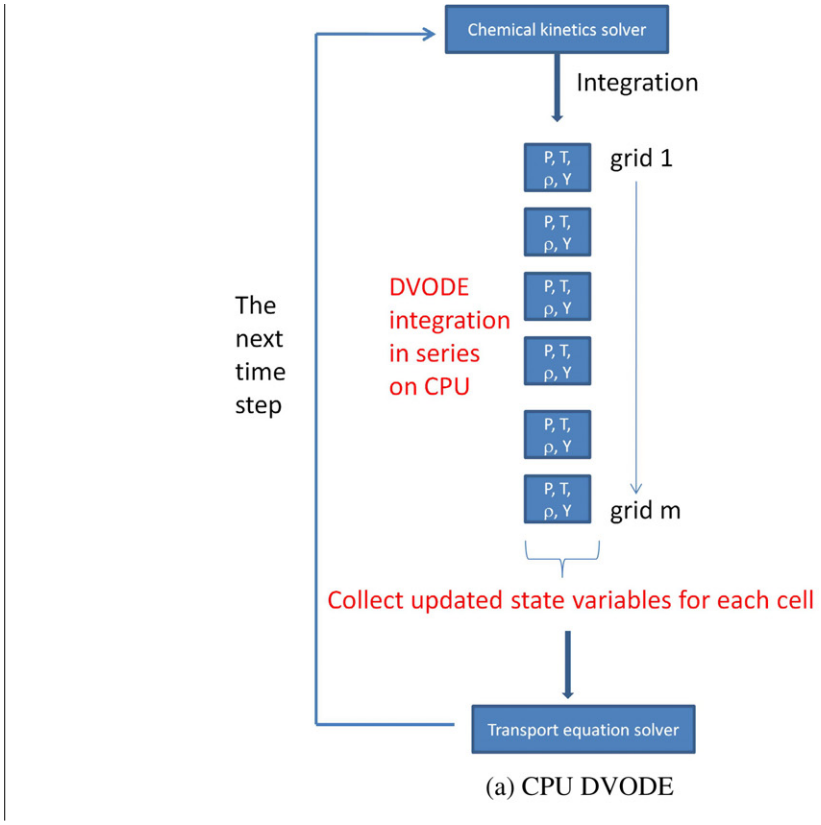
$$\tau_i = \min_i \begin{cases} a\frac{1}{p_i}, & \text{if } (0.1 \times \varepsilon \times q_i) > p_i y_i \\ a\frac{y_i}{|q_i - p_i y_i|}, & \text{otherwise} \end{cases}, \text{where } a = 0.5\sqrt{\varepsilon} \tag{8}$$

and subsequent time step sizes are determined using an adaptive time-stepping algorithm designed to minimize the frequency of violating Eq. (7) [8]. It is seen that in an ideal scenario, only two derivative function evaluations (predictor and corrector) are needed to complete each internal integration step. Detailed error analysis of the methods in CHEMEQ2 has been performed by Mott et al. [8]

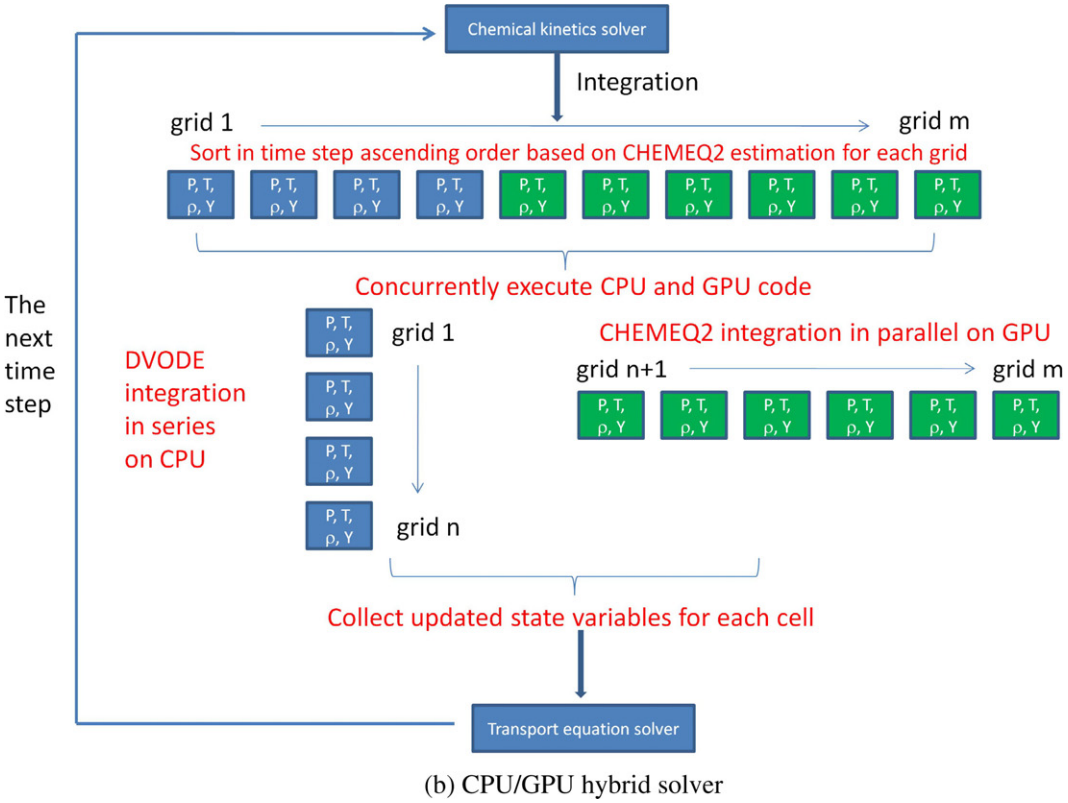
The original CHEMEQ2 solver considers every species in calculating time step size. We found that species with very small mass fractions may force the method to take extremely small time steps in some circumstances, particularly near equilibrium conditions. Qureshi and Prosser also saw similar behavior when using the original CHEMEQ2 solver in 1-D flame simulations and they concluded that this was due to the convergence criterion (Eq. (7)) [16]. So, they introduced an additional parameter (an absolute tolerance) to Eq. (7), to derive a modified convergence criterion of the form:

$$\|y_i^C - y_i^P\| \leqslant \varepsilon(y_i^C + \delta) \tag{9}$$

This modified criterion avoids the excessive time step reduction caused by minor species whose compositions are much smaller than δ, below which changes in composition are considered unimportant in the simulation. With designed numerical experiments, Qureshi and Prosser also showed that such modification with $\delta = 10^{-10}$ can greatly improve their solver's efficiency with minimal accuracy loss in 1-D flame simulations. We adopted this modifica-



(a) CPU DVODE



(b) CPU/GPU hybrid solver

Fig. 1. Comparison of serial CPU DVODE solver and the parallel hybrid solver for chemical kinetics integration in combustion modeling.

tion in CHEMEQ2-GPU and set $\delta = 10^{-12}$ for all simulations, which yielded improved algorithm efficiency without sacrificing accuracy in all our numerical experiments.

We have implemented the original Fortran CHEMEQ2 algorithm with the modifications described above as a CUDA C GPU kernel function. We note that CHEMEQ2-GPU represents a stand-

alone, general purpose ODE integrator that can be used for performing GPU-based, massively parallel numerical integration of large sets of ODEs. Here, our independent ODEs are kinetics equations at several grid points, which require several quantities that are usually computed using (CPU-based) CHEMKIN libraries. Therefore, we also rewrote several CHEMKIN II subroutines into CUDA C device functions, in order to calculate species construction and destruction rates on GPU cores. Porting CHEMKIN libraries and generating the ODEs on GPU cores also reduces the data transfer load between CPU and GPU. Only the values of the state variables at each grid point are passed through the bandwidth-limited Peripheral Component Interconnect (PCI) interface, which helps the overall efficiency of the GPU solver. It should be noted that the state variables at each grid point are stored in CPU memory. The CPU memory addresses of the stored species are mapped to the GPU, so that the size of CFD problems (number of grid points) that can be handled by CHEMEQ2-GPU is not limited by GPU memory but CPU memory. However, the GPU global memory size determines the largest size of reaction mechanism (number of species) that the GPU integrator can solve, because the sizes of the working arrays in the solver scale linearly with the size of the mechanism. We found that CHEMEQ2-GPU can handle mechanisms as large as 1000 species on the Tesla C2050 GPU card with 3 GB memory.

2.3.2. A hybrid explicit/implicit ODE solver approach

The stiffness of the kinetic ODE system in Eq. (1) is a function of the chemical state $(T, P, \mathbf{Y})_j$, which typically varies drastically across all the grid points and over the course of a combustion simulation. Explicit stiff ODE solvers are often more efficient than implicit solvers at moderately stiff or non-stiff conditions, while implicit solvers are more efficient when the kinetic ODE system is highly stiff. Therefore, the concept of the hybrid approach is to apply the GPU to massively parallelize the ODE integrations for the less stiff grid points using an explicit method (as implemented in CHEMEQ2-GPU), while the highly stiff cells are computed using implicit integration on the CPU. CUDA environment supports concurrent executions of CPU and GPU codes, which presents an opportunity for higher level CPU/GPU parallel computing. So, this hybrid ODE integration approach yields computational savings both from massive parallelization on the GPU for the less stiff grid points, as well as from concurrent integrations of both groups of "highly stiff" and "moderately stiff" grid points.

Since the objective is to minimize simulation time, grid points are dynamically designated as either "highly stiff" or "moderately stiff" based on an estimated stiffness ranking and CPU/GPU computational load-balancing. On each global (transport) integration time step, the grid points are ranked by degree of stiffness and a load balancing scheme is used to determine the proportion of the stiffest grid points that are to be assigned to CPU-based implicit integration, while the rest are integrated using massively parallel GPU-based explicit integration, as illustrated in Fig. 1b.

We have implemented the hybrid ODE solver approach within an internal combustion (IC) engine computational fluid dynamics (CFDs) simulator, KIVA-CHEMKIN [17,18]. The well-known DVODE [15] was used for CPU-based implicit integration and CHEMEQ2-GPU was used for GPU-based explicit integration. A load balancing scheme previously used for parallel CPU computing in message passing interface (MPI) environment [19] was applied to dynamically assign grid points to DVODE or CHEMEQ2-GPU. The particular methods used in our implementation are further described below, though the hybrid integration approach can be implemented using different combinations of implicit/explicit ODE solvers as well as other methods for stiffness ranking and CPU/GPU workload balancing.

2.3.3. Methods for stiffness ranking and CPU/GPU load balancing

There exist rigorous methods to quantify the stiffness of a set of ODEs, such as the computational singular perturbation (CSP) method [20]. However, these methods normally involve complex matrix eigenvalue analysis and thus are computationally expensive. Using these methods to rank the stiffness of grid points would introduce large overhead costs. Here, we have employed the CHEMEQ2 algorithm in Eq. (8) for stiffness ranking. Since this method only involves species rate calculation at a single set of conditions, it provides a very fast estimation of the stiffness of the grid points. Eq. (8) is evaluated at each grid point and stiffness is estimated to be inversely proportional to τ_i (i.e., stiffer grid points require smaller integration time steps). Therefore, the grid points are ranked in increasing order of τ_i and the top M^n_{CPU} of the grid points are assigned to DVODE (CPU-based implicit integration), while the rest are assigned to CHEMEQ2-GPU. The value of M^n_{CPU} is initialized at 1 (i.e., $M^0_{CPU} = 1$), so that at the start of the engine simulation (where conditions are expected to be minimally stiff), CHEMEQ2-GPU is used for all but one grid point. This also serves to initialize the CPU/GPU load balancing algorithm described next.

One of the critical issues in parallel computing is to balance computing load among available resources, in order to minimize processor idle time. Load balancing within the GPU is controlled by instruction scheduling algorithms embedded in GPU and CUDA environment. The load balancing algorithm used here is designed to achieve (almost) equal computation times on the GPU and the CPU for integrating the kinetics ODEs on a given global time step in the combustion simulation. Specifically, we estimate the expected computing speed of CPU (DVODE) and GPU (CHEMEQ2-GPU) integrators for the current global time step based on their performances on the previous time step:

$$S^n_{CPU} = \frac{M^{n-1}_{CPU}}{t^{n-1}_{CPU}}, \quad S^n_{GPU} = \frac{M^{n-1}_{GPU}}{t^{n-1}_{GPU}}, \tag{10}$$

where superscript n indicates the current transport time step, and S represents the computing speed of the CPU or GPU (i.e., number of grid points M computed per wall-clock second t during the previous global time step). Then the assignment of grid points for the current time step is determined as:

$$M^n_{CPU} = \frac{S^n_{CPU}}{S^n_{CPU} + S^n_{GPU}} M_{total}, \quad M^n_{GPU} = \frac{S^n_{GPU}}{S^n_{CPU} + S^n_{GPU}} M_{total} \tag{11}$$

Shi et al. [19] have previously demonstrated this scheme to be very effective for load balancing in parallel CPU computing, for an IC engine CFD code with chemical kinetics integration in MPI environment.

3. Results and discussion

To test the performance of CHEMEQ2-GPU and the hybrid ODE solver approach, multi-dimensional diesel engine simulations with premixed charge were conducted using KIVA-CHEMKIN. Table 1 summarizes the engine specifications and operating conditions. Three meshes representing coarse to fine mesh quality were created in order to test the scalability of the hybrid solver approach with the number of grid points, which are listed in Table 2 and illustrated in Fig. 2. In the simulation, the diesel combustion chemistry is represented using a surrogate fuel n-heptane. In this study, we compared the solvers' performances for two n-heptane mechanisms of different sizes. The small n-heptane mechanism has 39 species and 131 reactions [21], and the larger one contains 117 species and 499 reactions [22]. Both mechanisms include a NO_x sub-mechanism. The engine soot emission is calculated based on the soot precursor C_2H_2 species using a two-step modeling approach [23].

Table 1
Engine specifications and operating conditions.

Engine	GM-Fiat
Bore (mm)	82
Stroke (mm)	90.4
Compression ratio	16.6
IVC (aTDC)	−132
EVO (aTDC)	112
Speed (rev/min)	2000
Equivalence ratio	0.5
EGR	30%
IMEP (bar)	6.5

Table 2
Number of grids at bottom dead center (BDC) and top dead center (TDC) for three different engine meshes.

Mesh	Coarse	Medium	Fine
BDC	1700	4984	14,960
TDC	580	1344	4320

The simulation code was compiled using the Intel Fortran (v11.0) and CUDA GCC (v4.4.1) compilers on a 64-bit Ubuntu Linux PC with O2 level optimization. All simulations were performed on a PC with an Intel I7 930 quad-core CPU. The goal of these studies is to evaluate the effective number of additional CPU cores provided by CHEMEQ2-GPU, so the CPU-based codes were not parallelized and only utilized one core of the quad-core CPU. GPU computations were performed on a double-precision GPU card – NVIDIA Tesla C2050 which has 448 GPU cores and 3 GB memory. The simulation wall-clock times were recorded to compare the computational costs of the different solvers. In these analyses, the speedup of CHEMEQ2-GPU relative to CHEMEQ2 represents the number of (equivalent) *additional* CPU processors that CHEMEQ2-GPU provided in the simulation. The speedup of hybrid versus CHEMEQ2 represents the total number of equivalent CPU processors provided by the hybrid solver (noting that the simulation actually uses one physical CPU processor or "core").

Table 3 summarizes the total time spent on chemical kinetics integration of the engine combustion simulations using different ODE solvers for the three meshes. The flow solver time is included in Table 3 for comparison. The integration times are also plotted against the number of grids (grid points) of the three meshes in Fig. 3. Comparing the third and fourth row in Table 3, it is seen that the explicit CHEMEQ2 solver is consistently faster overall than the implicit DVODE solver. The time of the CPU-based solvers increases linearly with the number of grid points, as shown in Fig. 3. CHEMEQ2-GPU is about three times faster than the CPU-based CHEMEQ2 solver for the coarse mesh case, and thirteen times faster for the fine mesh case. This is also reflected in Fig. 3 which shows the simulation time of the GPU-based solver scales sub-linearly with the number of grid points. As extensively discussed in our previous study [5], the way that one can achieve high throughput in GPU computing is to compute an adequately large number of parallel elements to hide the overhead due to the communication cost. In the GPU based implicit ODE solver developed by the authors [5,6], the parallel elements refer to species and reactions. Therefore, we obtained better performance when applying that solver to larger reaction mechanisms. The same principle applies to the new solver in this study in which the parallel elements refer to grid points. This explains why CHEMEQ2-GPU becomes more efficient relative to CHEMEQ2 for more grid points. Furthermore, the hybrid solver which concurrently executes the CPU-based DVODE solver and CHEMEQ2-GPU shows the fastest performance. It is seen that for the coarse mesh case, the hybrid solver is 27% faster than CHEMEQ2-GPU. Compared to the DVODE solver, the hybrid solver

Table 3
Comparison of the time spent on chemical kinetics integration of each solver for the small mechanism.

	Coarse mesh	Medium mesh	Fine mesh
CPU based KIVA flow solver (min)	0.48	1.71	7.51
CPU based DVODE solver (min)	84.87	240.93	770.91
CPU based CHEMEQ2 solver (min)	33.11	85.59	308.76
GPU based CHEMEQ2 solver (min)	10.82	13.71	24.10
GPU/CPU hybrid solver (min)	7.85	11.19	20.57
Speed up of hybrid/DVODE solver	10.80	21.53	37.48
Speed up of hybrid/CHEMEQ2 solver	4.21	7.65	15.01

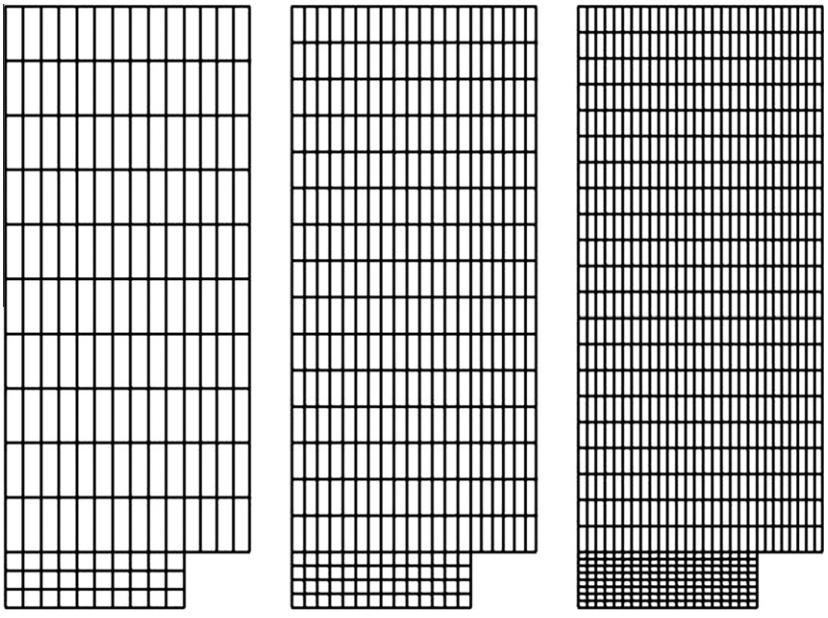


Fig. 2. Front view of the computational meshes, from left to right: Coarse, medium, and fine meshes.

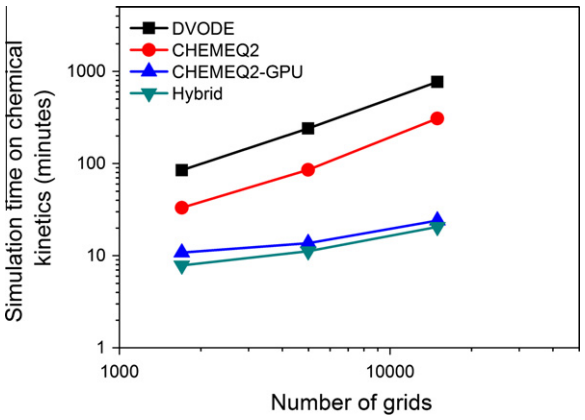


Fig. 3. Scaling of the solvers with the number of computational grids (finite volumes) for the small mechanism.

significantly speeds up the chemical kinetics integration by factors of 11, 22, and 37 for the three meshes, respectively. Compared to CHEMEQ2, the speedups are 4, 8, and 15-fold for the three meshes. CHEMEQ2-GPU also yields accurate results. Taking the fine mesh simulation for example, Fig. 4 shows that the predicted engine in-cylinder pressure trace, soot and NO_x emissions are very close for DVODE and CHEMEQ2-GPU. Although not shown here, we found that both major and minor species predicted using CHEMEQ2-GPU and the hybrid solver approach were very consistent with those predicted by the DVODE solver in all simulations. It is noted that this small discrepancy should not be a concern for the practical use of CHEMEQ2-GPU in multi-dimensional combustion modeling, given the fact that fluid dynamic calculations are seldom this accurate.

The wall-clock time spent on a global time step for the chemical kinetics integration using different ODE solvers is plotted as a function of engine crank angle in Fig. 5. It is also seen that during the main heat release stage (-13 to -10 °C A) CHEMEQ2 is much slower than DVODE, which indicates its poor performance for very stiff conditions. However, comparison of the time profiles of CHEMEQ2 and DVODE indicates that the QSS explicit method is more efficient than the BDF implicit method for the pre-ignition and post-combustion stages. Therefore, the overall computation time of CHEMEQ2 is still less than that of DVODE. Compared to CHEMEQ2, CHEMEQ2-GPU significantly reduces the computation time for each global time step as shown by the dotted curve in Fig. 5. For the coarse mesh case, CHEMEQ2-GPU is slower than DVODE during the main combustion stage, and this becomes comparable for the medium mesh case. For the fine mesh case, CHEMEQ2-GPU becomes faster than DVODE even in the main combustion stage. For pre-ignition and post-combustion conditions, CHEMEQ2-GPU is consistently much faster than DVODE, which saves a large amount of time. In general, the CHEMEQ2 algorithm based on the QSS explicit integration method shows different performances for stiff conditions and less stiff conditions, but the implicit DVODE solver behaves consistently during the entire simulation cycle. On a per-finite-volume basis, DVODE is more efficient than CHEMEQ2 for the finite volumes with highly stiff chemistry.

Gou et al. [24] also observed the same behavior of explicit and implicit integration methods, which motivated the development of a dynamic multi-timescale method for chemical kinetics integration. In their method, species are categorized into fast and slow groups based on their instantaneous time scales during kinetics integration, and the fast and slow groups are integrated with implicit and explicit methods, respectively. They reported an order of magnitude speed-up using this hybrid approach. In our approach,

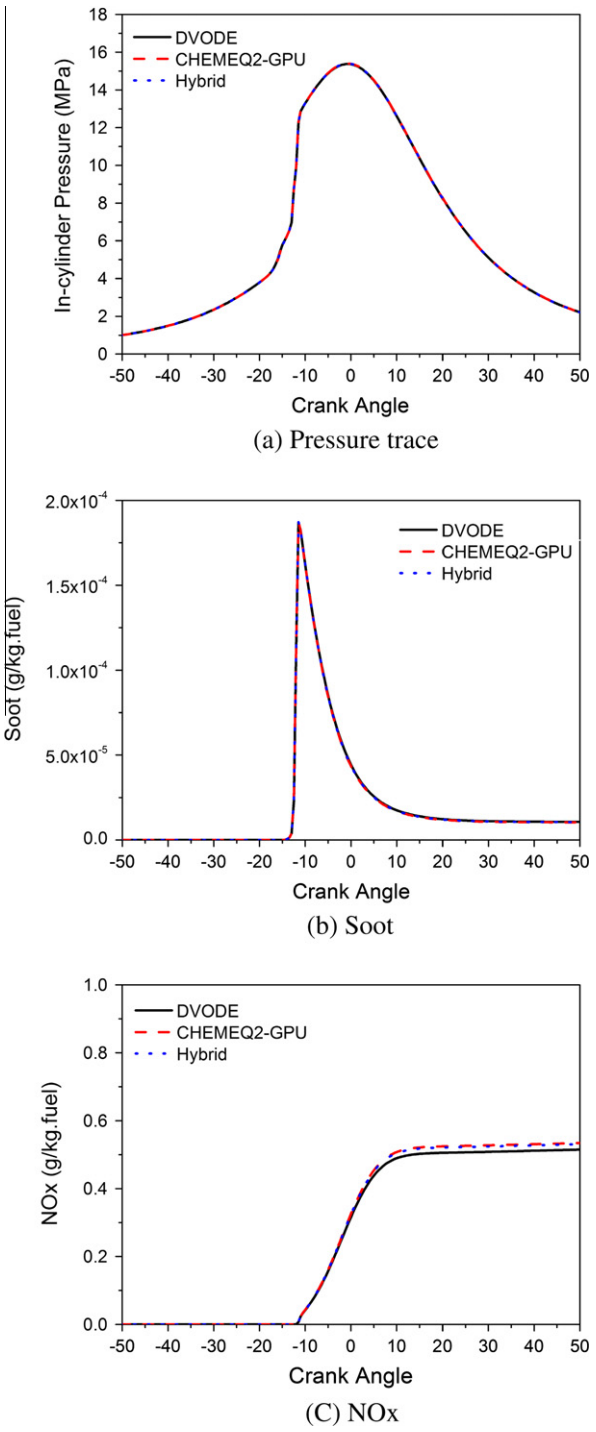


Fig. 4. Comparison of simulation results computed by the CPU based DVODE solver and the hybrid solver for the fine mesh case with the small mechanism.

grid points are separated into two groups based on their stiffness. The stiffest grid points are integrated using the CPU-based implicit solver (DVODE) and the less stiff grid points are computed using the GPU-based explicit CHEMEQ2 solver. In Fig. 5, the time profiles of this hybrid CPU-based implicit/GPU-based explicit solver (represented by the dash-dotted curves) are all below those of CHEMEQ2-GPU solver, demonstrating the further speed-up.

To better examine the performance of the load balancing scheme, the wall-clock time profiles of the CPU-based and GPU-based solvers are plotted in Fig. 6. It is observed that the times

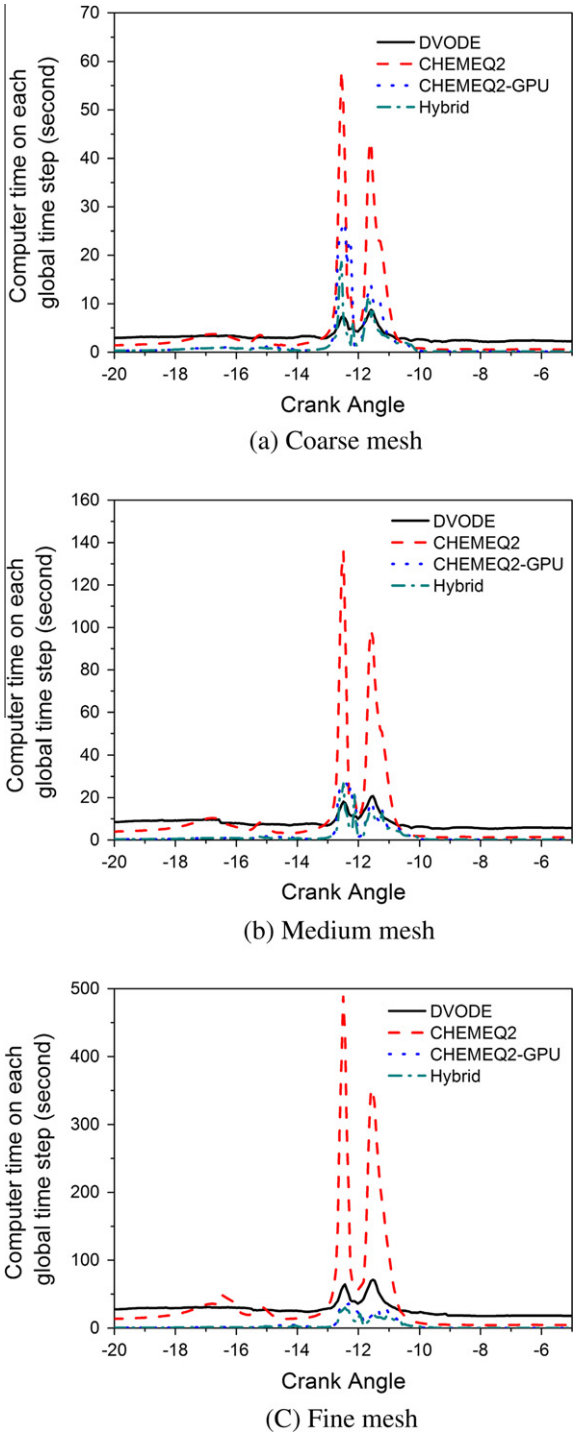


Fig. 5. Comparison of time profile of the ODE solvers.

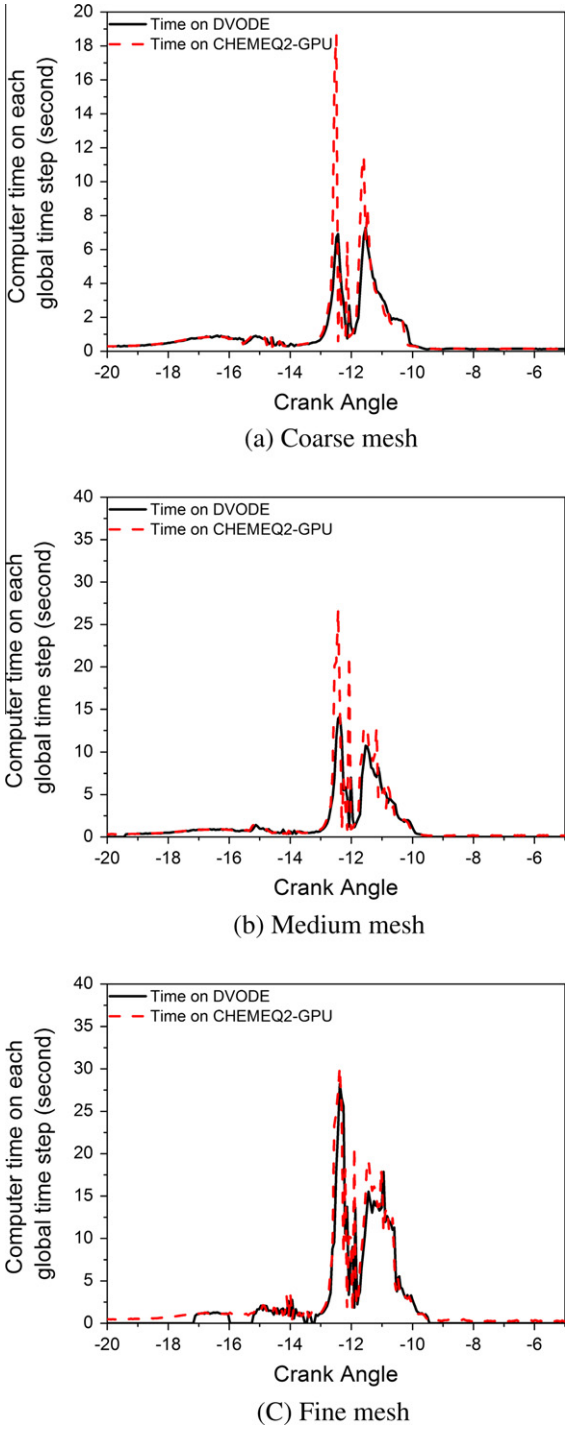


Fig. 6. Comparison of time profile of the DVODE solver and the CHEMEQ2-GPU solver in the hybrid solver.

spent in the CPU and GPU solvers are very close at most crank angles, indicating the effectiveness of the present load balancing scheme. The only exception is that during the transition from less stiff to stiffer conditions, the load balancing scheme fails to assign a proper number of grid points to each solver, which results in higher load to CHEMEQ2-GPU. This is probably due to a lag in response caused by the purely feed-back control nature of the load balancing algorithm applied. However, this lag lasts for only a few global time steps and does not have a noticeable impact on the overall efficiency of the hybrid solver.

The engine simulations were repeated using the large reaction mechanism, which has three times as many species as the small mechanism (117 versus 39 species). Table 4 lists the time spent on kinetics integration using different ODE integrators and the time spent on the flow solver. Compared to the times listed in Table 3, the integration times of both DVODE and CHEMEQ2 scale quadratically with the number of species in the kinetic mechanism. The CHEMEQ2-GPU and hybrid solvers scale slightly less than quadratically. For the fine mesh case, the hybrid solver is about 46 times faster than DVODE, and 14 times faster than CHEMEQ2.

Table 4
Comparison of the time spent on chemical kinetics integration of each solver for the large mechanism.

	Coarse mesh	Medium mesh	Fine mesh
CPU based KIVA flow solver (min)	0.63	2.15	10.61
CPU based DVODE solver (min)	702.43	1920.60	6339.83
CPU based CHEMEQ2 solver (min)	216.73	539.23	1868.36
GPU based CHEMEQ2 solver (min)	75.54	88.51	141.91
GPU/CPU hybrid solver (min)	59.40	85.92	138.56
Speed up of hybrid/DVODE solver	11.83	22.35	45.76
Speed up of hybrid/CHEMEQ2 solver	3.65	6.28	13.48

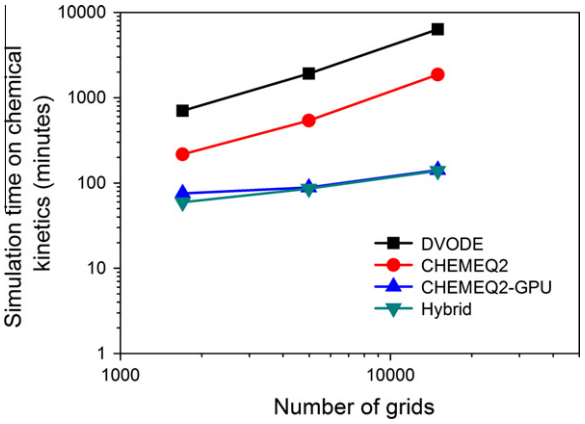


Fig. 7. Scaling of the solvers with the number of computational grids (finite volumes) for the large mechanism.

Figure 7 shows that mechanism size has no impact on how the computation times scale with the number of grid points.

Similar to Fig. 4, Fig. 8 compares the simulation results of engine pressure and emissions which show the consistency among the DVODE, CHEMEQ2-GPU and hybrid solvers. The time profiles for these cases are almost the same as in the cases with the small mechanism, so we do not include these comparisons here.

4. Summary and concluding remarks

We have described CHEMEQ2-GPU, a new explicit stiff ODE solver that exploits the parallel computing capabilities of the modern graphics processing unit (GPU) to accelerate multi-dimensional combustion simulations. We have also described a hybrid solver approach for deriving additional savings by applying a dynamic combination of CPU-based implicit and GPU-based explicit numerical integration algorithms. Our hybrid solver approach couples CHEMEQ2-GPU and an existing CPU-based implicit solver, DVODE. A dynamic load balancing scheme was used to assign the kinetics ODE integrations over all grid points to either the CPU or GPU, exploiting the robustness of DVODE for highly stiff grid points and the speed of CHEMEQ2-GPU for the grid points that are less stiff. We demonstrated CHEMEQ2-GPU and the hybrid solver for 3-D homogeneous charge compression ignition (HCCI) engine simulations, using two n-heptane reaction mechanisms of different sizes and three meshes (ranging from 1700 to 14,960 grid points at engine BDC). Assuming ideal linear scaling of simulation time with number of processors, the speed of CHEMEQ2-GPU on the Tesla C2050 GPU was equivalent to CHEMEQ2 running on approximately 13 parallel 2.8 GHz CPU processors; and the hybrid solver approach was equivalent to CHEMEQ2 on ~15 such CPU processors. In summary, CHEMEQ2-GPU provided the additional computing power of 14 parallel CPU processors and the hybrid solver approach demonstrated a method to efficiently apply these additional co-processors with existing CPU cores for combustion simulations. CHEMEQ2-GPU is available by request to the authors.

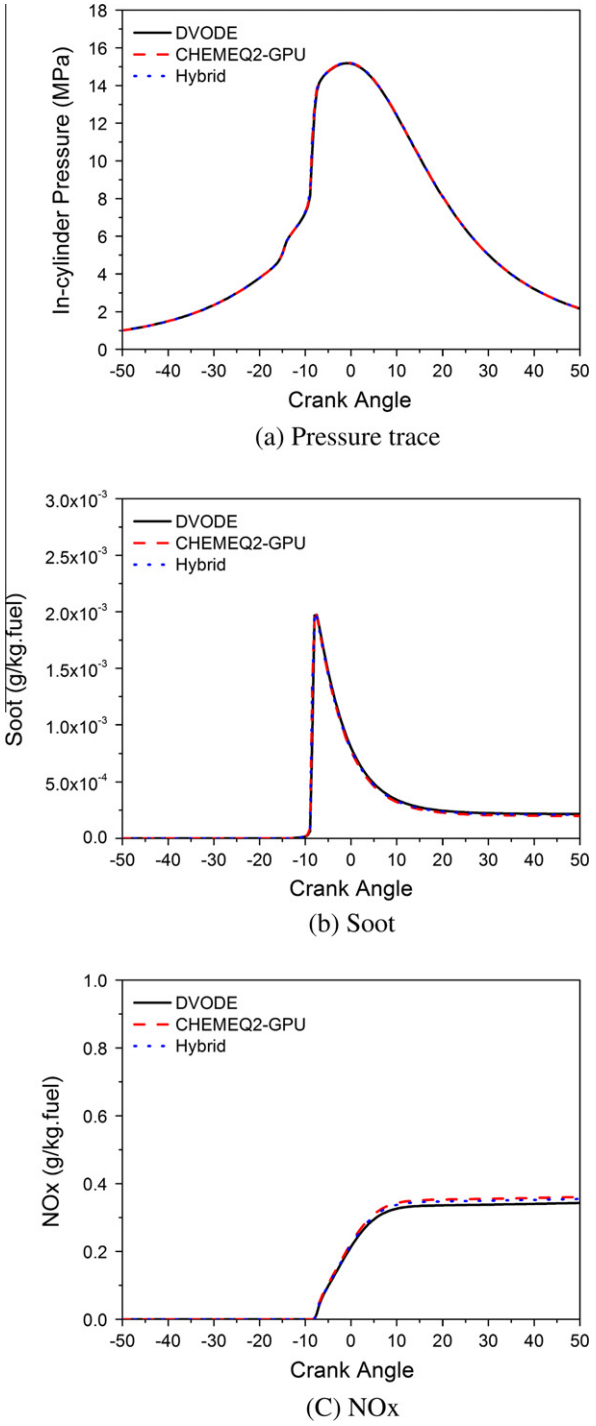


Fig. 8. Comparison of simulation results computed by the CPU based DVODE solver and the hybrid solver for the fine mesh case with the large mechanism.

Several opportunities exist for improving the hybrid algorithm as presented here, most notably the methods for stiffness ranking and load balancing. In this study, we used a simple, fast method based on the integration time step estimation to separate highly stiff and moderately stiff grid points for implicit and explicit solvers. The simple method was effective for HCCI conditions investi-

gated here. However, more sophisticated and efficient methods may be required for other combustion conditions. In our preliminary test studies with direct injection engine simulation (inhomogeneous combustion with more highly stiff grid points during the main combustion stage), we observed less speedup relative to DVODE alone using the same hybrid solver (typically limited to about a factor of 3 speedup) – largely due to reduced effectiveness of the CPU/GPU (implicit/explicit ODE integration) load balancing algorithm under these conditions. Therefore, our ongoing work is focused on developing improved methods for stiffness ranking and load balancing. Importantly, we note that the speedup of CHEMEQ2-GPU relative to CHEMEQ2 (on a given CPU and GPU) is not dependent on the problem stiffness, but only on the number of grid points in the problem. So, even in the DI engine simulations, CHEMEQ2-GPU still yields the equivalent of up to ~14 additional parallel CPU processors (cores) running CHEMEQ2 for similar meshing as was used in the HCCI cases.

Acknowledgments

We thank the National Aeronautics and Space Administration (NASA) Small Business Innovation Research (SBIR) Program under Contract Number NNX11CD84P and the Combustion Energy Frontier Research Center (CEFRC) of the US Department of Energy (DOE) for supporting this work. The authors also want to thank Dr. Raymond Speth at MIT for his invaluable comments on the code development.

References

[1] T. Lu, C.K. Law, Prog. Energy Combust. Sci. 35 (2009) 192–215.
[2] J. Chen, A. Choudhary, B. de Supinski, M. De Vries, E.R. Hawkes, S. Klasky, W.K. Liao, K.L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, C.S. Yoo, Comput. Sci. Discovery 2 (2009) 015001.
[3] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, J.C. Phillips, Proc. IEEE 5 (2008) 879–899.
[4] NVIDIA CUDA. <http://www.nvidia.com/object/cuda_home_new.html> (accessed 2011).
[5] Y. Shi, W.H. Green, H.-W. Wong, O.O. Oluwole, Combust. Flame 158 (2011) 836–847.
[6] Y. Shi, O.O. Oluwole, H.-W. Wong, W.H. Green, A multi-approach algorithm for enabling efficient application of very large, highly detailed reaction mechanisms in multi-dimensional HCCI engine simulations, in: 7th US National Technical Meeting of the Combustion Institute, Georgia Institute of Technology, Atlanta, March 2011.
[7] K. Spafford, J. Mercedith, J. Vetter, J. Chen, R. Grout, R. Sankaran, Accelerating S3D: a GPGPU case study, in: Seventh Int. Workshop HeteroPar, 2009.
[8] D.R. Mott, E.S. Oran, L.V. Bram, J. Comput. Phys. 164 (2000) 407–428.
[9] NVIDIA CUDA Programing Guide 3.2, 2011.
[10] I.S. Ufimtsev, T.J. Martinez, J. Chem. Theory Comput. 4 (2008) 222–231.
[11] J.E. Stone, J.C. Phillips, P.L. Freddolino, D.J. Hardy, L.G. Trabuco, K. Schulten, J. Comput. Chem. 28 (2007) 2618–2640.
[12] J.C. Linford, J. Michalakes, M. Vachharajani, A. Sandu, IEEE Trans. Parallel Distrib. Syst. 22 (2011) 119–131.
[13] Z. Ren, S. Pope, J. Comput. Phys. 227 (2008) 8165–8176.
[14] D.A. Schwer, P. Lu, W.H. Green, V. Semiao, Combust. Theory Model. 7 (2003) 383–399.
[15] P.N. Brown, G.D. Byrne, A.C. Hindmarsh, SIAM J. Sci. Stat. Comput. 10 (1989) 1038–1051.
[16] S.R. Qureshi, R. Prosser, Implementation of α-QSS stiff integration methods for solving the detailed combustion chemistry, in: Proceedings of the World Congress on Engineering, vol. II, London, UK, July 2007.
[17] A.A. Amsden, KIVA-3V: A Block-structured KIVA Program for Engines with Vertical or Canted Valves, Los Alamos National Laboratory, Report No. LA-13313-MS, 1997.
[18] S.-C. Kong, C.D. Marriott, R.D. Reitz, Modeling and Experiments of HCCI Engine Combustion Using Detailed Chemical Kinetics with Multidimensional CFD, SAE Paper 2001-01-1026, 2001.
[19] Y. Shi, S.L. Kokjohn, H.-W. Ge, R.D. Reitz, Efficient Multidimensional Simulations of HCCI and DI Engine Combustion with Detailed Chemistry, SAE Paper 2009-01-0701, 2009.
[20] S.H. Lam, D.A. Goussis, Int. J. Chem. Kinet. 26 (1994) 461–486.
[21] A. Patel, S.-C. Kong, R.D. Reitz, Development and Validation of a Reduced Reaction Mechanism for HCCI Engine Simulations, SAE Paper 2004-01-0558.
[22] Y. Ra, R.D. Reitz, Combust. Flame 158 (2011) 69–90.
[23] S.-C. Kong, Y. Sun, R.D. Reitz, J. Eng. Gas Turb. Power 129 (2007) 245.
[24] X. Gou, W. Sun, Z. Chen, Y. Ju, Combust. Flame 157 (2010) 1111–1121.