Contents lists available at ScienceDirect

Computers & Fluids

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

GPU acceleration for general conservation equations and its application to several engineering problems

Fang-An Kuo [a,1], Matthew R. Smith [a,*], Chih-Wei Hsieh [a,2], Chau-Yi Chou [a,3], Jong-Shinn Wu [b]

[a] *National Center for High-performance Computing, HsinChu, Taiwan*
[b] *Department of Mechanical Engineering, National Chiao Tung University, HsinChu, Taiwan*

ARTICLE INFO

Article history:
Received 30 August 2010
Received in revised form 27 September 2010
Accepted 12 October 2010
Available online 16 October 2010

Keywords:
Finite volume method
Graphics Processing Unit (GPU)
Computational Fluid Dynamics (CFD)
Vector-split fluxes
Parallel computation

ABSTRACT

Presented is a general method for conservation equations called SHLL (split HLL) applied using Graphics Processing Unit (GPU) acceleration. The SHLL method is a purely vector-split approximation of the classical HLL method (Harten et al., 1983 [1]) which assumes the presence of local wave propagation in the algebraic derivation of fluxes across cell surfaces. The conventional HLL flux expression terms are interface-split and wave propagation velocities estimated (where required) based on local conditions. Due to the highly local nature of the SHLL fluxes, the scheme is very efficiently applied to GPU computation since the flux, initialisation and update phases of the computation are all vectorized processes. The SHLL scheme is applied to GPU computating using Nvidia's CUDA package. Numerical schemes are presented for solutions to the general transport (convection–diffusion) equation, Euler Equations and Shallow Water Equations with results presented for several benchmark gas and shallow water flow engineering problems. Computational times are compared between high-end GPU (Nvidia C1060) and CPU (Intel Xeon 3.0 GHz, 32 MB cache) systems with reported speedups of over 67 times when applied to two dimensional simulations with multi-million cell numbers.

1. Introduction

Conventional application of numerical algorithms to the high-performance simulation of scientific or engineering related problems has conventionally relied upon the Multiple Instruction stream, Multiple Data stream (MIMD). In this approach, a machine has a large number of processors which operate asynchronously and independently on varying sets of information as supplied by the programmer. While this approach permits a large degree of flexibility on the user, in many instances, the algorithm applied to parallel computation is almost identical to that employed in serial (single CPU) computation – a clear indication that this flexibility is typically not well exploited by the programmer. Very little modification of the original algorithms themselves are typically performed in order to optimize the parallel computation.

The recent rise (or one might say, return) to vectorized computation in the form of acceleration by Graphics Processing Units (GPUs) has prompted a rethink in computational strategy. The recent rise in computer gaming and demand for high performance graphics on commonly available PC's has lead to the development of relatively cheap, though highly efficient, computation platforms (video, or graphics, cards). These cards are designed to perform a large number of relatively simple computations in support of their primary purpose – drawing an image on the computer monitor. These computations, most typically performed on vectorized sets of data, more closely follows the Single Instruction stream, Multiple Data stream (SIMD) approach. Each set of data (and its instructions) are typically independent of other data streams, reducing communication between computational "threads" and allowing very rapid computation.

Applications of these powerful devices to scientific computation has recently risen in popularity amount the scientific community. This rise in popularity has not gone unnoticed by video accelerator manufacturers (such as Nvidia and ATI) and now a large number of devices are available which are dedicated to computation. These devices, an example being the Nvidia TESLA C1060, are similar to their gaming cousins in that computation should be approached from an SIMD perspective. This has made the porting of traditional numerical schemes, which are heavily reliant on the MIMD approach, to GPU computation very difficult. Due to the large amounts of communications between threads, communications between the host and device and heavy reliance upon globally

* Corresponding author. Tel.: +886 6 5576085x637.
E-mail addresses: mathppp@nchc.org.tw (F.-A. Kuo), msmith@nchc.org.tw (M.R. Smith), davidhsieh@nchc.org.tw (C.-W. Hsieh), cychou@nchc.org.tw (C.-Y. Chou), chongsin@faculty.nctu.edu.tw (J.-S. Wu).
[1] Tel.: +886 3 5776085x347.
[2] Tel.: +886 3 5776085x269.
[3] Tel.: +886 4 2462 0202x828.

0045-7930/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compfluid.2010.10.007

available variables, reported speedups on early GPU based codes have been quite low. As scientists continue to invest in GPU based computation, there is an emerging realisation that conventional parallelisation approaches are insufficient and new perspectives must be embraced.

Presented is the SHLL (split HLL) method, a numerical method which embraces the above perspective. The conventional HLL approach, developed by Harten et al. [1], is an algebraic method which computes fluxes across cell interfaces by assuming the presence of two propagating waves and integration of the governing Partial Differential Equation (PDE) over space–time. The original HLL scheme is ideal for serial computation and a traditional MIMD philosophy: however, the scheme is not ideal for vectorized computation. The presented SHLL scheme is the simple and straightforward modification of the HLL fluxes to vector-split form. The resulting expressions posses a higher degree of locality than the original HLL expressions at the expense of the conventional estimate of wave speeds used in the flux computation. The SHLL scheme is derived and applied to various conservation equations and finally employed for the simulation of several non-trivial engineering problems. The resulting speed-ups based on comparison between several high-end GPU and CPU architectures (Nvidia TESLA C1060 and the Intel Xeon 8× core systems respectively) are shown to be on the order of 55× (and higher) for simulations employing upwards of a million cells.

2. Harten, Lax and Van Leer (HLL) method

The fluxes developed by Harten et al. [1] are presented here in their complete integral form. Fig. 1 shows a control volume in $x-t$ space covering the region between cells i and $i+1$ centered on the interface separating the cells at $x = 0$. The region is temporally bound by the limits $t = 0$ and $t = T$. At $t = 0$, waves moving at velocities S_L (<0) and S_R (>0) move away from the discontinuity between the cell interface. The conditions inside the control volume in the region between $[0, T]$ and $[x_L, x_R]$ can be described by the integral:

$$\int_{S_L T}^{S_R T} U_*^T\,dx = \int_{x_L}^{0} U_L^0\,dx + \int_0^{x_R} U_R^0\,dx + \int_0^T F_L\,dt - \int_0^T F_R\,dt - \int_{x_L}^{S_L T} U_L^T\,dx - \int_{S_R T}^{x_R} U_R^T\,dx \tag{1}$$

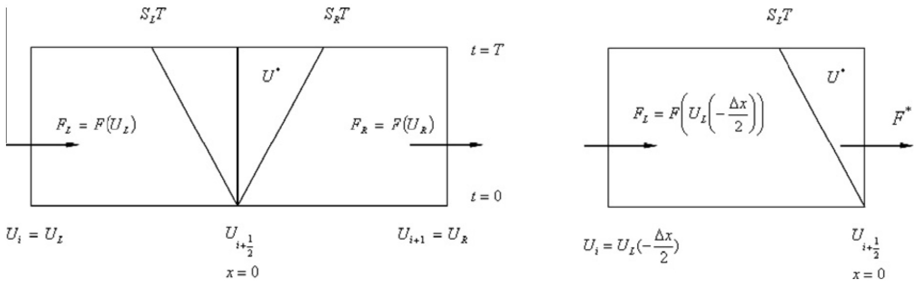
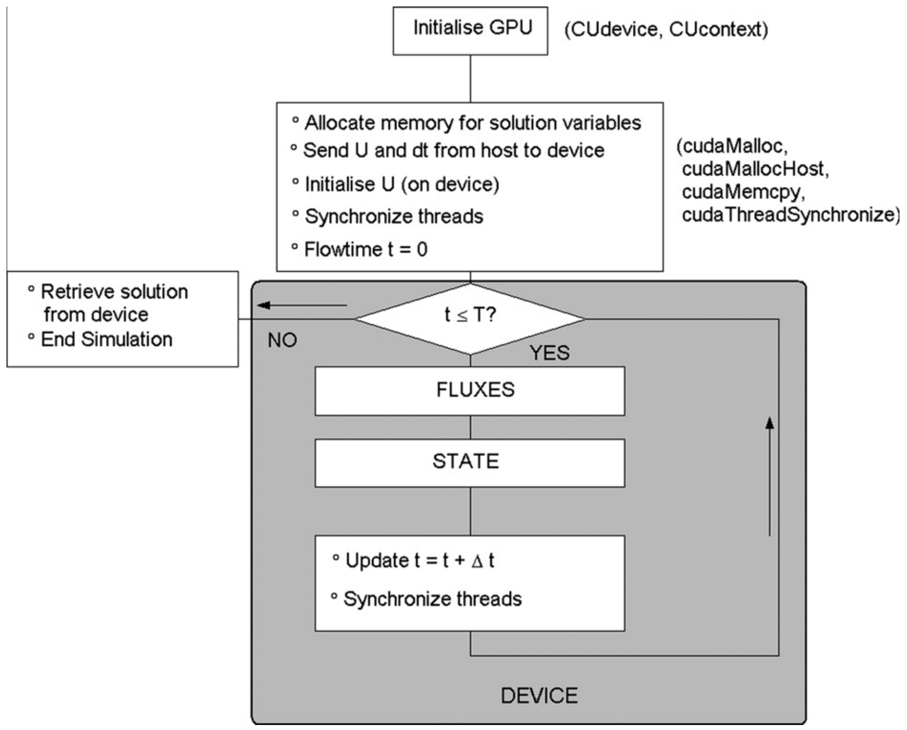
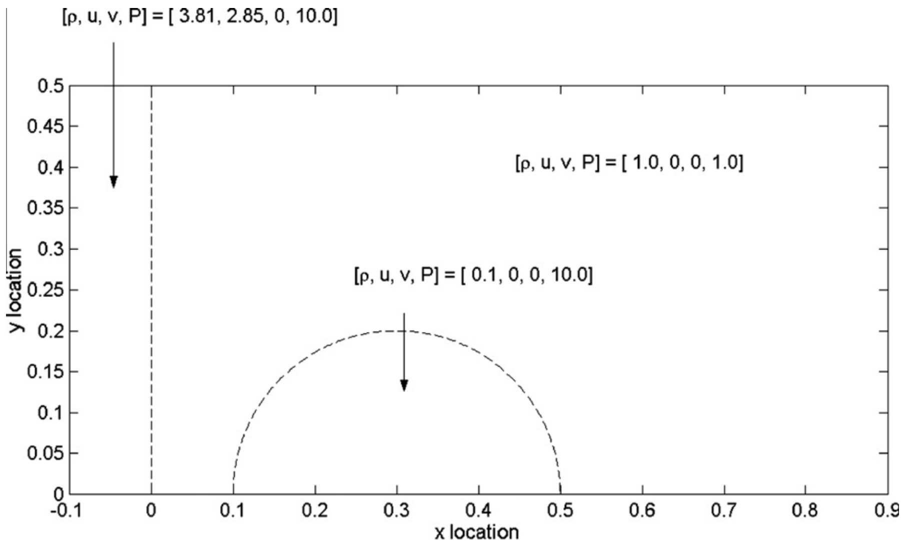


Fig. 1. [Left] Control Volume (CV) centered between left and right cell centres showing both propagating waves S_L and S_R [Right] CV centered on left-hand cell showing flux across star region F^*.



Initialise GPU
(CUdevice, CUcontext)
° Allocate memory for solution variables
° Send U and dt from host to device
° Initialise U (on device)
° Synchronize threads
° Flowtime t = 0
(cudaMalloc, cudaMallocHost, cudaMemcpy, cudaThreadSynchronize)
° Retrieve solution from device
° End Simulation
t ≤ T?
NO
YES
FLUXES
STATE
° Update t = t + Δ t
° Synchronize threads
DEVICE

Fig. 2. Implementation of SHLL using GPU. The kernels FLUXES() and STATE() are device-based, leaving the CPU free to manage communications between nodes in the event that multiple GPU devices are employed.



[ρ, u, v, P] = [3.81, 2.85, 0, 10.0]
[ρ, u, v, P] = [1.0, 0, 0, 1.0]
[ρ, u, v, P] = [0.1, 0, 0, 10.0]
y location
x location

Fig. 3. Problem configuration for the 2D shock-bubble interaction problem. The top and bottom surfaces are reflective. The left and right hand boundaries are inflow and outflow surfaces respectively.

where the subscripts L and R represent conditions in the left and right cells respectively, while the superscript 0 and T represents conditions at time $t = 0$ and $t = T$ respectively. The subscript $*$ represents conditions in the *star region* between the propagating waves. This equation assumes nothing regarding variation of fluxes F or conserved quantities U within space and time. The only assumptions made are in the presence of the two propagating waves surrounding a single intermediate region at $x = 0$. Fig. 1 shows the revised $x - t$ diagram focusing on the left cell only. Using the same method of integrating conserved quantities over space and fluxed quantities over time obtains:

$$\int_{x_L}^{0} U_L^0\,dx + \int_0^T F_L\,dt - \int_0^T F_* = \int_{x_L}^{S_LT} U_L^T\,dx + \int_{S_LT}^{0} U_*^T\,dx \tag{2}$$

By assuming that the average state over the region between $x = S_LT$ and $x = S_RT$ is the same as the average between the region $x = S_LT$ and $x = 0$, we can substitute the equations together to obtain the expression for the interface flux:

$$\begin{aligned}\int_0^T F_* &= \int_{x_L}^{0} U_L^0\,dx - \int_{x_L}^{S_LT} U_L^T\,dx + \int_0^T F_L\,dt - \int_{S_LT}^{0} U_*^T\,dx \\ &= \int_{x_L}^{0} U_L^0\,dx - \int_{x_L}^{S_LT} U_L^T\,dx + \int_0^T F_L\,dt \\ &\quad + \frac{S_L}{S_R - S_L}\left(\int_{x_L}^{0} U_L^0\,dx - \int_{x_L}^{S_LT} U_L^T\,dx + \int_0^{x_R} U_R^0\,dx\right. \\ &\quad \left. - \int_{S_RT}^{x_R} U_R^T\,dx + \int_0^T F_L\,dt - \int_0^T F_R\,dt\right)\end{aligned} \tag{3}$$

By assuming that the quantities U remain spatially constant (i.e. monotonic in nature) and the fluxes are temporally constant we recover the original HLL flux expressions:

$$\overline{F_*} \approx \frac{F(U_L)S_R - F(U_R)S_L - S_RS_L(U_R - U_L)}{S_R - S_L} \tag{4}$$

3. SHLL (split HLL) method

The mathematical vector splitting of the HLL expressions is a trivial exercise: the HLL flux expressions are separated into LHS and RHS components:

$$F = F^+ + F^- = \frac{F_L - S_LU_L}{\left(1 - \frac{S_L}{S_R}\right)} + \frac{F_R - S_RU_R}{\left(1 - \frac{S_R}{S_L}\right)} \tag{5}$$

where superscripts + and – indicate forward and backward moving fluxes across the interface under investigation. However, these expressions require computation involving conditions on opposite sides of the interface – a luxury we cannot afford if we are to successfully split the equations. Instead, were required, we will replace the correct wave speeds with an approximate value:

$$F = \frac{F_L - S_LU_L}{\left(1 - \frac{S_L}{S_R^A}\right)} + \frac{F_R - S_RU_R}{\left(1 - \frac{S_R}{S_L^A}\right)} \tag{6}$$

where superscript A indicates an approximate estimate of the correct value. The simplest approximate estimate of the correct wave speeds might be $S_R^A \approx V_L + a_L$ and $S_L^A \approx V_R - a_R$. After substitution, the complete split fluxes are of the form:

$$\begin{aligned} F^+ &= F_L\left(\frac{\chi_L + 1}{2}\right) + U_La_L\left(\frac{1 - \chi_L^2}{2}\right) \\ F^- &= -F_R\left(\frac{\chi_R - 1}{2}\right) - U_Ra_R\left(\frac{1 - \chi_R^2}{2}\right)\end{aligned} \tag{7}$$

where χ_L and χ_R are the critical speeds (i.e. Mach number for Eulerian gas flow, Froude number for Shallow Water flow) on the left and right hand sides respectively. These fluxes posses a desirable feature: when $\chi_R = \chi_L = 1$, the net flux is F_L and likewise, when $\chi_R = \chi_L = -1$, the net flux is F_R. It should be clear that this scheme is applicable to any set conservative partial differential equations (i.e. conservation equations) where the solutions permit flow moving with bulk velocities V and characteristic disturbances (waves) propagating with speeds a. The resulting net flux equation can also be written in terms of the central difference flux:

$$F = \frac{1}{2}(F_L + F_R) - \frac{1}{2}\{F_R\chi_R - F_L\chi_L + U_Ra_R(1 - \chi_R^2) - U_La_L(1 - \chi_L^2)\} \tag{8}$$

which demonstrates the schemes reductions to the Rusanov flux [2] when the critical numbers reduce to 0. The latter term may be regard as an artificial diffusion term – the addition of a dissipation coefficient D applied to the latter term would allow one to reduce the numerical dissipation. The scheme can now be demonstrated

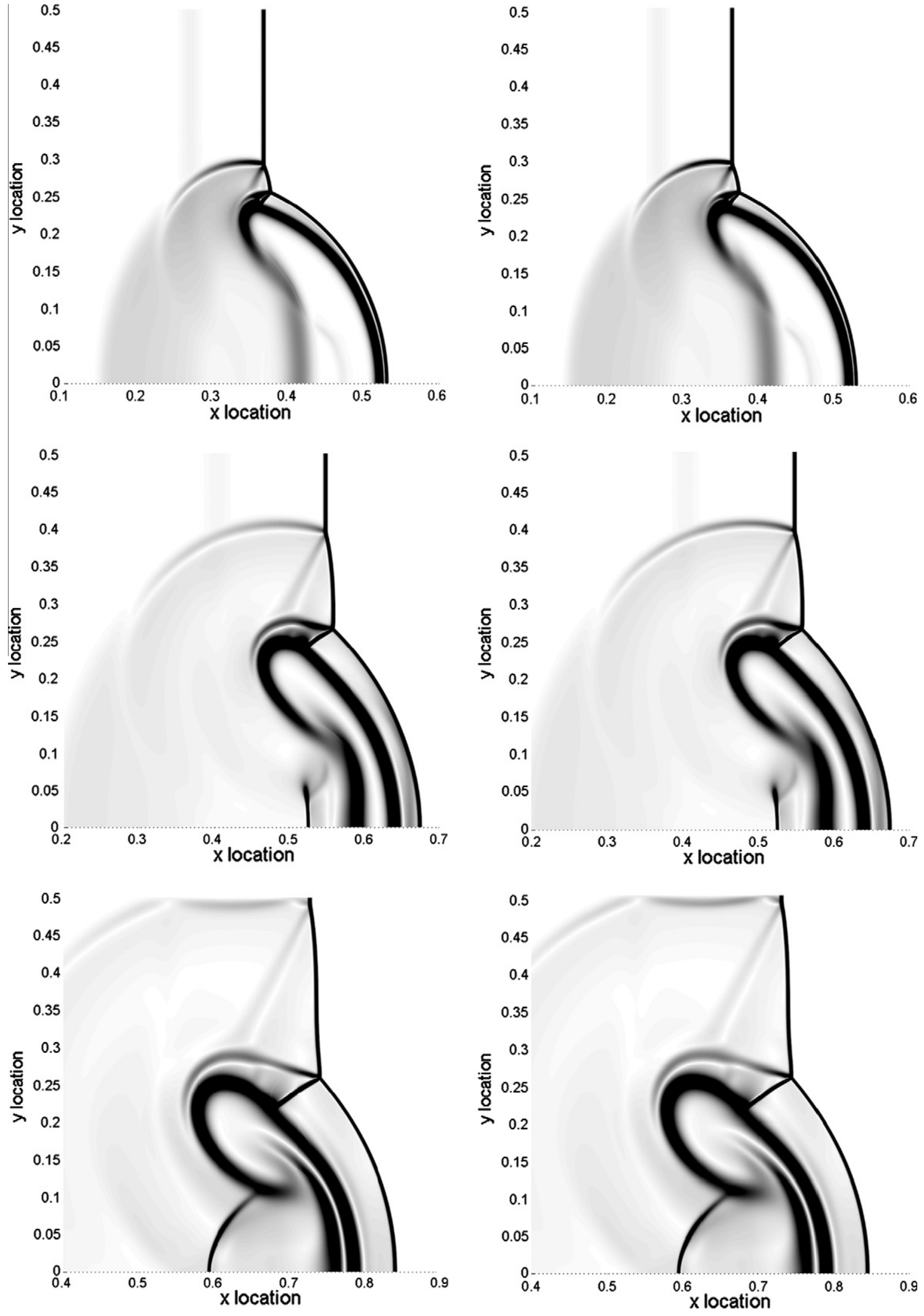


Fig. 4. Results from [Left] SHLL and [Right] HLL for the first order accurate simulation of the 2D shock-bubble interaction problem. Results are shown after flow time $t = 0.1$ [Top], $t = 0.15$ [Middle] and 0.2 [Bottom].

to be a central difference scheme with artificial dissipation controlled by the value of D:

$$F^+ = F_L\left(\frac{D\chi_L+1}{2}\right) + DU_La_L\left(\frac{1-\chi_L^2}{2}\right) \quad F^- = -F_R\left(\frac{D\chi_R-1}{2}\right) - DU_Ra_R\left(\frac{1-\chi_R^2}{2}\right) \tag{9}$$

4. Application to GPU computation

The SHLL method is accelerated through application of GPU computation as shown in Fig. 2. Prior to GPU kernel execution, all of the relevant data must be transferred from the host PC to the GPU device. Due to the limited bandwidth between the two and the large amount of data transferred, this process has the potential to severely limit performance and thus should be avoided as much as possible. To increase performance, data transfer is reduced by running the entire simulation on the GPU device without interacting with the host PC. The three main functions present on the GPU device in vectorized form are:

- Initialization kernel – the computational grid is generated and the initial density, velocity and temperature assigned. This process is made simpler through application of Cartesian grids.
- Flux kernel – The split fluxes to the adjacent neighbouring cells are computed.
- State kernel – each cell receives/donates each respective incoming/outgoing split flux and then updates the values of conserved properties (i.e. mass, momentum and energy). Following this, the primitives (density, velocity and temperature) along with gradients of conserved properties are computed.

In the situation of a two dimensions problem, we create m blocks and n threads for parallel computation in a CUDA kernel where m is equal to the number of cells in the y-dimension and n is equal to the number of cells in the x-dimension. Due to the limited number of threads, if the number of x-dimension is more than 512 then n is set to 512 and one (or more) of these threads have to handle multiple data. Cacheable texture memory is used in our CUDA implementation: an efficient form of memory which is capable of reading source split flux data from global memory. Texture memory is accessed within multiple kernels through using such device functions as ‘tex1Dfetch’, called *texture fetches*. A texture fetch only counts as a device memory read in the instance of a cache memory miss. During the flux kernel execution primitive variables (which are constant during kernel execution) are read through texture fetching. Threads of like warp which read these texture addresses (which are close together) have been observed as achieving excellent performance. Such fetches are also designed for streaming with constant latency.

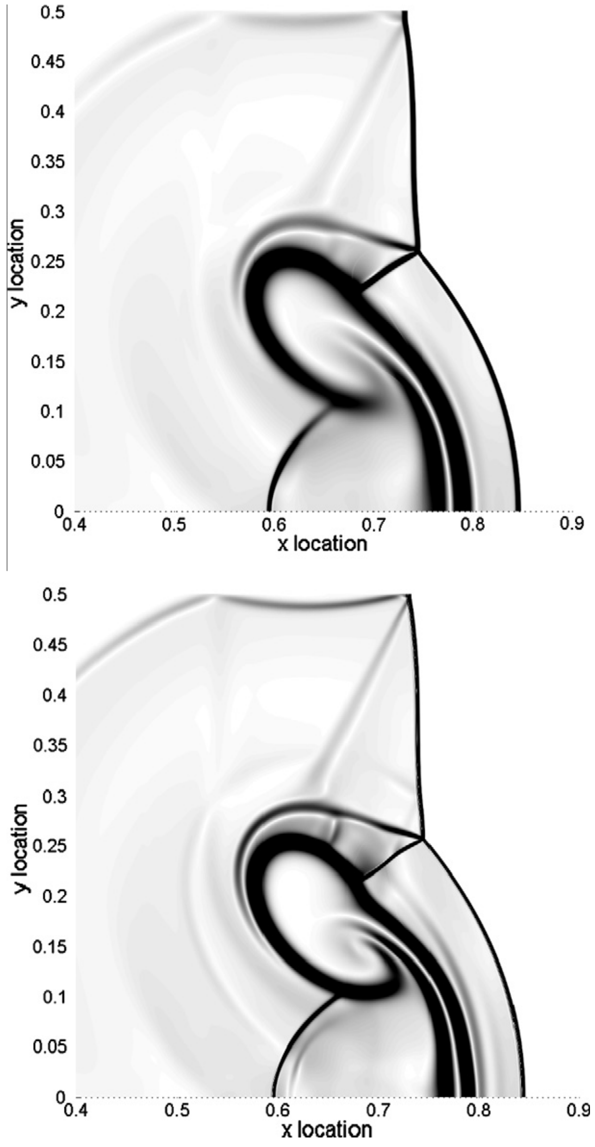


Fig. 5. Solutions to the 2D shock-bubble interaction problem using SHLL with $D = 1$ [Top] and $D = 0.5$ [Bottom]. Results are shown at flow time $t = 0.2$. Note the clearly reflected wave from the top boundary and its increased resolution with reduced dissipation coefficient.

5. Results

5.1. Euler Equations – 2D shock/bubble interaction problem

The two dimensional Euler Equations are applied to the simulation of a shock wave passing through an artificial “bubble” [3] as shown in Fig. 3. The two dimensional Euler Equations are:

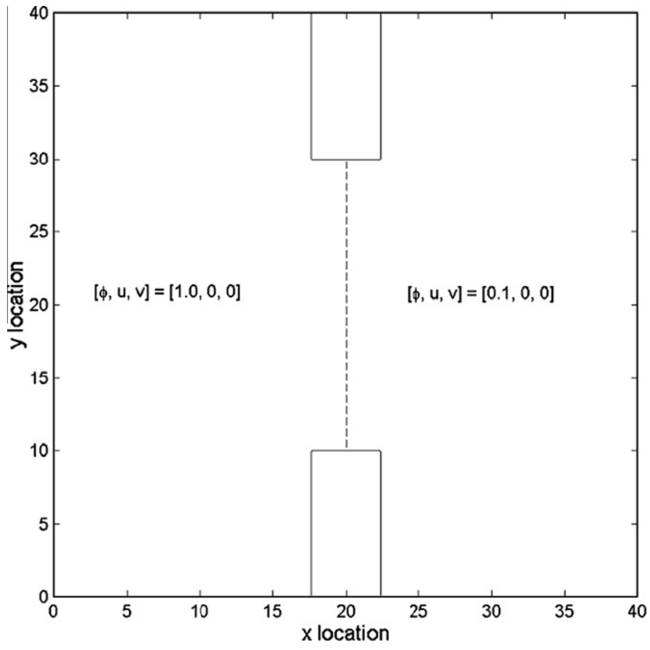


Fig. 6. Problem geometry for the 2D shallow-water dam break benchmark. The finite thickness of the dam walls are $w = 1/8L$.

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ E\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ u(E+p)\end{bmatrix}+\frac{\partial}{\partial y}\begin{bmatrix}\rho v\\ \rho vu\\ \rho v^2+p\\ v(E+p)\end{bmatrix}=0 \qquad (10)$$

where ρ is the gas density, u and v are velocities in the x and y directions respectively and p is the pressure computed from the ideal gas law ($p = \rho RT$). The energy E is given by $E = \rho(0.5\,V^2 + C_vT)$ where C_v is the specific heat constant at constant volume and T is the temperature. The characteristic propagation speed is the local speed of

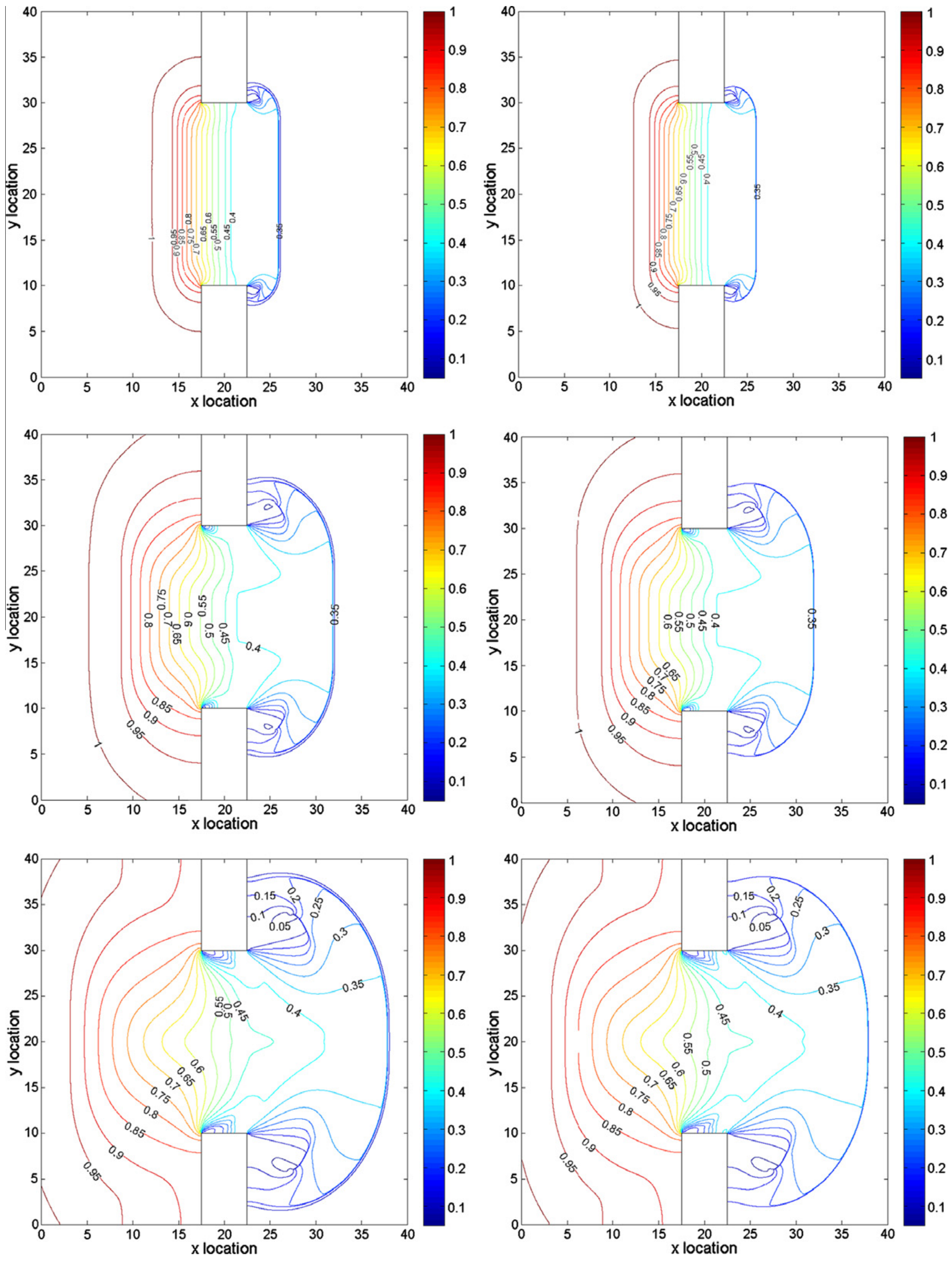


Fig. 7. Contours of geopotential taken from [Left] SHLL and [Right] SHLL for the 2D inviscid shallow-water dam break problem after flow time $t = 6$ [Top], 12 [Middle] and 18 [Bottom]. All simulations have used 2048 × 2048 cells with a constant maximum CFL of 0.25.

sound $a = (\gamma RT)^{0.5}$. The flow is advanced until flow time $t = 0.2$. All simulations employed a constant CFL number of 0.5. The simulations are fixed at first order accuracy in both space and time to clearly demonstrate the influence of the splitting and of the dissipation coefficient D.

Presented in Fig. 4 are numerical schlierens (indicative of gradients of density) demonstrating the difference between the conventional HLL and SHLL results. It is worth noting that these significant differences arise only due to the difference in estimated wave speeds S_L and S_R, which affect numerical dissipation. The influence of the value of dissipation coefficient D on the results for the two dimensional shock-bubble interaction problem is demonstrated in Fig. 5. Reduction of the dissipation coefficient to $D = 0.5$ reveals flow features previously unseen using conventional SHLL or first order accurate HLL flux expressions. The clearer depiction of the high-pressure circulation located at $\sim(x, y) = (0.65, 0.15)$, in addition to numerous other flow features, clearly demonstrates the reduced dissipation.

5.2. Inviscid Shallow Water Equations – 2D dam break problem

An ideal two dimensional dam break problem is considered as shown in Fig. 6. The governing equations employed are the Shallow Water Equations, defined as:

$$\frac{\partial}{\partial t}\begin{bmatrix}\phi\\ \phi u\\ \phi v\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\phi u\\ \phi u^2+\frac{\phi^2}{2}\\ \phi uv\end{bmatrix}+\frac{\partial}{\partial y}\begin{bmatrix}\phi v\\ \phi uv\\ \phi v^2+\frac{\phi^2}{2}\end{bmatrix}=0 \tag{11}$$

where $\Phi = gh$, g is gravitation acceleration ($\approx$9.81) and h is the water level. The values u and v are the velocity in the x and y directions respectively. At $t = 0$, the invisible barrier separating the high and low water levels is removed and the flow allowed to develop until $t = 20$ s. The resulting contours of Φ at times $t = 10$ and $t = 18$ s is shown in Fig. 7. The comparison between solutions shows almost identical results due to the lack of a contact surface in the star region, lowering the scheme's consequent dependence on estimations of S_L and S_R. All solutions are deliberately fixed at first order accuracy in time and space for comparison between the split and unsplit form of HLL.

6. Parallel performance

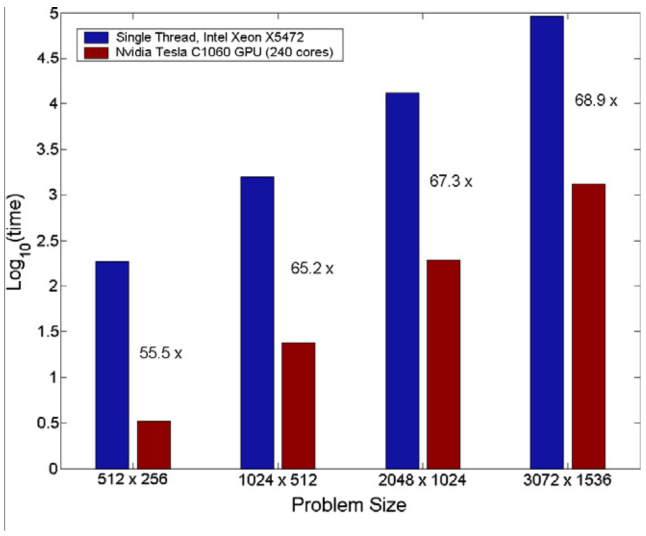
The efficiency of the scheme is only slightly dependent on the numerical equations being solved, while being much more dependent on the problem size. This is demonstrated in Tables 1 and 2 upon examination of performance characteristics for solutions of the Euler and Shallow Water Equations by SHLL. Additional comparison can be found in Fig. 8. For low numbers of cells (i.e. 100 × 100 cell simulations), the initialization time and communication times associated with GPU computation reduce the speedup (defined as the ratio of computational time required by a single CPU and a GPU computation) to approximately unity. However, for larger problems (i.e. 1000 × 1000 cells), the speedup was mea-

Table 1
Computational expense (in seconds) for the solution of a 2D shock-bubble interaction problem solved using SHLL on both a single CPU and a single Nvidia C1060 GPU computing device for problems of varying computational complexity (i.e. size).

	512 × 256	1024 × 512	2048 × 1024	3072 × 1536
CPU time (s) Intel Xeon X5472 3 GHz, 12 MB L2 cache	184.52	1568.69	13066.183	91,180
GPU time (s) Nvidia Tesla C1060 240 × cores (1.44 GHz)	3.32	24.07	194.12	1322.9
Speedup (CPU/GPU)	**55.55**	**65.16**	**67.31**	**68.923**

Table 2
Computational expense (in seconds) for the solution of the 2D shallow-water dam break problem solved using SHLL on both a single CPU and a single Nvidia C1060 GPU computing device for problems of varying computational complexity (i.e. size).

	256 × 256	512 × 512	1024 × 1024	2048 × 2048
CPU time (s) Intel Xeon X5472 3 GHz, 12 MB L2 Cache	18.948	155.08	1247.76	9951.15
GPU time (s) Nvidia Tesla C1060 240 × cores (1.44 GHz)	0.41	2.68	19.95	158.32
Speedup (CPU/GPU)	**46.56**	**57.84**	**62.55**	**62.85**



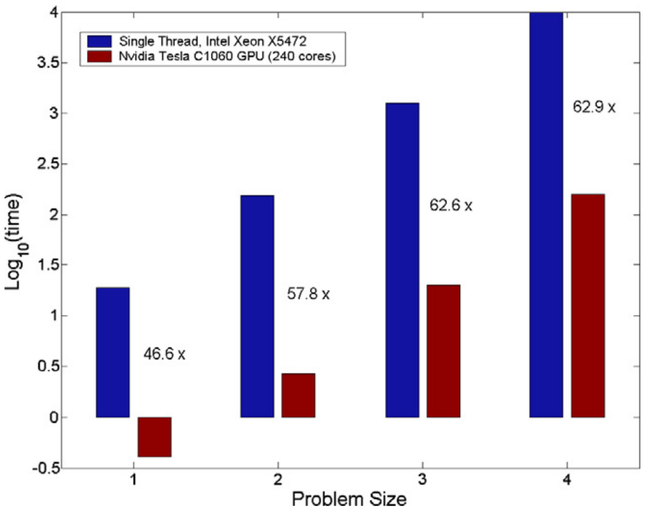


Fig. 8. Computational expense $\log_{10}$(time in seconds) for [Top] 2D Euler Equations and [Bottom] 2D Shallow Water Equation benchmarks of varying problem size (i.e. computational cells). Results compare a single thread of an Intel Xeon X5472 3.0 GHz processor against the computational time required by an Nvidia Tesla C1060 GPU computing device for an identical problem configuration. A difference of two (2) between logarithmic times represents a speedup ratio of 100 times (i.e. 10^2 = 100).

sured to be >62×. This speedup could be improved by moving a fraction of the workload from the GPU back to the CPU (host), however, the exact fraction and tasks to be performed by the host fall outside the scope of this investigation, focussing on pure GPU computation only.

7. Conclusion

The Harten, Lax and van Leer (HLL) flux has been recast as a split flux method (SHLL) for the efficient application to GPU computation. The increased locality of the resulting split scheme results in more efficient computation owing to the reduction in communications required for flux computation. The investigation of an additional "dissipation coefficient" is entered after comparison of the SHLL fluxes to the Rusanov flux, showing a slight improvement in the numerical dissipation present. Solutions are presented for both inviscid gas flow (Euler Equations) and inviscid shallow water flows. When accelerated using a single Nvidia C1060 GPU computing device, the performance increases over 60-fold when compared to the performance on a single thread of an Intel X5472 Xeon CPU (3.0 GHz, 12 MB cache). The best specific time (i.e. time per cell per time step) of the implementation is approximately 3.5×10^{-9} s per cell per time step for the two dimensional benchmarks investigated here.

References

[1] Harten A, Lax PD, van Leer B. On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. SIAM Rev 1983;25(1):35–61.
[2] Rusanov VV. Calculation of intersection of non-steady shock waves with obstacles. USSR J Comput Math Math Phys 1961;1:267–79.
[3] Cada M, Torrilhon M. Compact third-order limiter functions for finite volume methods. J Comput Phys 2009;228(11):4118–45.