Contents lists available at ScienceDirect

Computer Physics Communications

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

Parallel implementation of geometrical shock dynamics for two dimensional converging shock waves

Shi Qiu, Kuang Liu, Veronica Eliasson [*]

Aerospace and Mechanical Engineering, University of Southern California, Los Angeles, CA 90089-1191, USA

ARTICLE INFO

Article history:
Received 2 February 2016
Received in revised form
18 May 2016
Accepted 8 June 2016
Available online xxxx

Keywords:
Geometrical shock dynamics
Parallel computing
Converging shock
Symmetric boundary conditions

ABSTRACT

Geometrical shock dynamics (GSD) theory is an appealing method to predict the shock motion in the sense that it is more computationally efficient than solving the traditional Euler equations, especially for converging shock waves. However, to solve and optimize large scale configurations, the main bottleneck is the computational cost. Among the existing numerical GSD schemes, there is only one that has been implemented on parallel computers, with the purpose to analyze detonation waves. To extend the computational advantage of the GSD theory to more general applications such as converging shock waves, a numerical implementation using a spatial decomposition method has been coupled with a front tracking approach on parallel computers. In addition, an efficient tridiagonal system solver for massively parallel computers has been applied to resolve the most expensive function in this implementation, resulting in an efficiency of 0.93 while using 32 HPCC cores. Moreover, symmetric boundary conditions have been developed to further reduce the computational cost, achieving a speedup of 19.26 for a 12-sided polygonal converging shock.

1. Introduction

Focusing of shock waves can generate extreme conditions, such as high pressure and temperature, at the focal region. Shock focusing occurs in a variety of man-made and naturally occurring events, for example in extracorporeal shock wave lithotripsy [1], inertial confinement fusion [2] and collapse of cavitation bubbles [3]. The first researcher to study shock focusing was Guderley [4], who developed analytical solutions for converging cylindrical and spherical shock waves. Following his work, numerous authors have investigated on this area. Basically, there are two major numerical methods used to study shock wave propagation: one is the Navier–Stokes equations, and the other is the inviscid Euler equations if viscosity in the shocked medium can be neglected. The advantage of these two methods is that a full flow field can be accurately obtained. However, the strength of the shock front in the focusing area can be much higher than that of the initial shock front, resulting in smaller time scales to maintain the Courant–Friedrichs–Lewy (CFL) condition. In addition, the length of the shock front in the focusing area can be much smaller than that of the initial shock front. In order to resolve all small scales close to, and in, the focal region, high resolution in both time and space are required, which can make the computational task very expensive. Whitham proposed an alternative method [5], named Geometrical Shock Dynamics (GSD), that describes the motion of shock waves in a different way. Unlike the Navier–Stokes equations and the Euler equations, this theory avoids dealing with the flow field around the shock and only focuses on the curvature of the shock wave itself. As a result, solving a shock focusing event with GSD instead of the Navier–Stokes equations or the Euler equations, the computational complexity is reduced by solving a lower dimensional problem. In addition, the actual computational cost may be reduced by more than an order of magnitude depending on the required grid resolution when dealing with higher dimensional problems. In GSD, the shock front is discretized into small elements. Between each element, orthogonal trajectories are introduced as rays so that each shock front element can be approximated to propagate down a tube whose boundaries are defined by the rays, a so-called ray tube. The main assumption in GSD is that the motion of the shock only changes with the variation of the ray tube area. Then, instead of solving the full Euler equations, the motion of the shock can be predicted by deriving the relation between shock strength, which can be represented by the Mach number, M, and the area of the ray tube, A. This is the so-called Area-Mach

* Corresponding author.
E-mail address: eliasson@usc.edu (V. Eliasson).

http://dx.doi.org/10.1016/j.cpc.2016.06.003
0010-4655/© 2016 Elsevier B.V. All rights reserved.

number (A–M) relation,

$$\frac{1}{A(x)}\frac{dA}{dM} = -g(M), \tag{1}$$

where

$$g(M) = \frac{M}{M^2-1}\left(1+\frac{2}{\gamma+1}\frac{1-\mu^2}{\mu}\right)\left(1+2\mu+\frac{1}{M^2}\right), \tag{2}$$

$$\mu^2 = \frac{(\gamma-1)M^2+2}{2\gamma M^2-(\gamma-1)}. \tag{3}$$

Here, γ represents the adiabatic index, x denotes the distance in the ray tube and the cross sectional area $A(x)$ is a function of x, see Ref. [6] for additional details.

There exists a variety of algorithms to implement GSD numerically. Here, only a part of those works are listed. The method of characteristics was used by Bryson and Gross [7] to analyze shock diffractions. Decades later, a front tracking approach was developed by Henshaw [8] in two dimensions and Schwendeman [9] in three dimensions. A conservative finite difference method was formulated by Schwendeman [10]. Most recently, a fast-marching like algorithm was developed by Noumir et al. [11]. Among all these algorithms, the front tracking method is the most used one due to its computational accuracy and simplicity of implementation. For example, Schwendeman applied this method to study shock motion in non-uniform media [12]. Best modified it to investigate underwater explosions [13,14], and Apazidis and Lesser utilized it to compare with the shock converging experiments [15]. However, in order to utilize the front tracking method to study shock motion for large scale applications, the computational cost is still a large bottleneck, which can be addressed through parallel computing techniques.

In our work, the front tracking method has been implemented on parallel computers and symmetric boundary conditions has been developed in order to reduce the computational expense dramatically.

2. Numerical methods

2.1. Serial scheme

In this study, the numerical implementation of GSD is based on previous front tracking methods [8,13,14]. A short review about implementation of the numerical scheme is given as follows. In a two dimensional condition, the shock front is discretized into N points denoted by $\boldsymbol{x_i}(t)$, where $i = 1, 2, \ldots, N$. The shock front propagates along the direction of its normal vector, and the speed is determined by the A–M relation. The motion of the shock is described by

$$\frac{d\boldsymbol{x_i}(t)}{dt} = a_0 M_i(t)\boldsymbol{n_i}(t), \quad i = 1, \ldots, N, \tag{4}$$

where $M_i(t)$ and $\boldsymbol{n_i}(t)$ denote the Mach number and shock front normal at $\boldsymbol{x_i}(t)$, respectively. The speed of sound in ambient air is fixed as $a_0 = 1$ in all of our calculations. A fourth-order Runge–Kutta scheme is adopted to numerically integrate equation (4).

The Mach number, $M(t)$, is calculated by combining the A–M relation from Eq. (1) with the shock front relation $d_tA = a_0 M d_x A$. Thus, Eq. (1) can be converted into

$$\frac{dM}{dt} = \frac{M}{-g(M)}\frac{A'}{A}. \tag{5}$$

In Eq. (5), A' denotes d_xA and A'/A can be expressed explicitly as [13],

$$\frac{A'}{A} = \frac{\partial\boldsymbol{x}(s(t),t)}{\partial s(t)}\frac{\partial\boldsymbol{n}(s(t),t)}{\partial s(t)}, \tag{6}$$

where, as is shown below, the arclength $s(t)$ represents the geometry of the shock front and $\boldsymbol{n}(s(t),t)$ indicates the norm vector of the shock front,

$$s_i(t) = \begin{cases} 0, & \text{if } i = 1; \\ s_{i-1}(t) + |\boldsymbol{x_i}(t) - \boldsymbol{x_{i-1}}(t)|, & \text{if } i = 2, \ldots, N; \end{cases} \tag{7}$$

$$\boldsymbol{n}(s(t),t) = \left(\frac{\partial y(s(t),t)}{\partial s(t)}, -\frac{\partial x(s(t),t)}{\partial s(t)}\right). \tag{8}$$

Following [8], there are two extra procedures. One is a point insertion and deletion approach that maintains the resolution of the shock front and the CFL condition. Additionally, a time-step reduction scheme [16] should be applied to replace this approach for a particular converging shock scenario. The other scheme is a two-step smoothing procedure to reduce the high frequency errors. The main focus of this study is to implement the algorithm for converging shocks. Thus, here the time-step reduction scheme is adopted. However, the parallel implementation can be coupled with the point insertion and deletion approach to tackle other shock wave simulations.

The time-step reduction scheme can be expressed by the following equation:

$$\Delta t < f(R(t), M(t)), \tag{9}$$

where $f(R(t), M(t))$ is a function of shock radius $R(t)$ and $M(t)$ at time t. More details of this scheme can be found in [16]. If this condition fails, Δt will be reduced as $\Delta t_{new} = 0.75\Delta t$ and Eqs. (4) and (5) are restarted by using Δt_{new}.

The two-step smoothing procedure is given as follows:

$$\boldsymbol{x_i}(t) = \frac{1}{2}(\boldsymbol{x_{i-1}}(t) + \boldsymbol{x_{i+1}}(t)), \tag{10}$$

and it is applied every n_s iterations, where n_s depends on the time increment, Δt, and the average shock arclength between each discrete point, Δs_{avg}. When $\Delta s_{avg} = 0.01$, n_s is usually set between 10 and 50.

A schematic flow chart shown in Fig. 1 illustrates how this algorithm works. In general, the algorithm consists of four functions during the time marching process. At time t, A_i'/A_i is calculated first by Eq. (6). The two components $\partial\boldsymbol{x}(s(t),t)/\partial s(t)$ and $\partial\boldsymbol{n}(s(t),t)/\partial s(t)$ in Eq. (6) can be obtained by applying a cubic spline interpolation method using discrete data $s_i(t), \boldsymbol{x_i}(t)$ and $s_i(t), \boldsymbol{n_i}(t), i = 1, \ldots, N$ under adequate boundary conditions, which vary according to the task settings. This step is named *updateAda*. The advantage of setting *updateAda* initially is that it can also be used to generate the norm vector of the shock front, see Eq. (8), which is required for updating the shock location in the third step. Next, a fourth-order Runge–Kutta scheme is employed, see Eq. (5), to update the Mach number at the current time iteration. This step is named *updateM*. In the next function, called *updateX*, the shock location for the next time iteration is calculated by using Eq. (4) so that $\boldsymbol{x}(t + \Delta t)$ is obtained. Later, the time-step reduction scheme is applied to maintain the CFL condition, see Eq. (9). The next function, *updateS*, computes $s(t + \Delta t)$ through Eq. (7). In addition, the smoothing procedure, see Eq. (10), is implemented after *updateS*. At the end of each iteration, a terminate condition is given to determine whether the program should be aborted. In order to run this algorithm, a set of coordinates, $\boldsymbol{x_i}(t_0)$, and Mach number, $M_i(t_0)$, representing the initial location and strength of the shock front are given as initial conditions. In addition, *updateX* and *updateS* need to be called before the time marching to obtain $s_i(t_0+\Delta(t))$, which is the arclength for the first iteration. These initial steps are implemented in the parameter initialization, shown in Fig. 1.

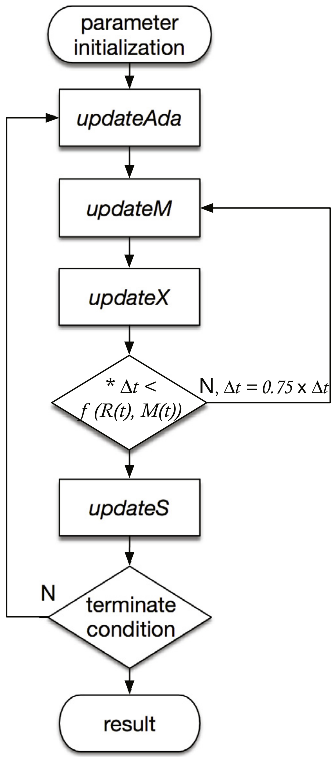

Fig. 1. Flow chart for serial version. The equation to maintain the CFL condition is marked by $*$.

2.2. Parallel scheme

Performance profiling has been done to detect the computational cost for each function described in the previous section. The test setup is a 10-sided polygonal (decagon) shape shock wave propagating towards its center point. Three different cases have been tested, see Table 1. The Mach number, smooth step interval and the initial shock arclength between each discrete point are kept constant as $M_i(t_0) = 15, n_s = 200, \Delta s(t_0) = 0.002$ for all cases while the apothem of the decagon (a line segment from the center to the midpoint of one of its sides) varies from 1 to 100. All three cases are terminated after 10,000 iterations. Fig. 5 shows an example when the apothem equals 1 and the program terminates after 55,000 iterations. The blue solid polygons represent the initial and the first reconfiguration steps, which will be explained in the validation section. The red solid polygons represent the successive shock fronts at each iteration sequence.

Table 1 shows the computational costs of each function in different cases. It can be seen that most of the computations are conducted by the first four functions, occupying over 98% of the total running time. And *updateAda* is the most expensive one, which occupies over 64% of the total running time. The reason is that to solve Eq. (6), $\partial \boldsymbol{x}(s(t), t)/\partial s(t)$ needs to be obtained first by applying four cubic spline functions since periodic boundary conditions are used. Then, $\boldsymbol{n}(s(t), t)$ can be achieved through Eq. (8) automatically. Later, another four cubic spline functions are employed to access $\partial \boldsymbol{n}/(s(t), t)\partial s(t)$, where $\boldsymbol{n} = (n_x, n_y)$. Thus, there are eight tridiagonal systems of linear equations in total that have to be solved because each cubic spline function can be considered as a tridiagonal system. Commonly, Gaussian elimination can solve linear systems in a serial manner. However, for parallelization purpose, it is not efficient compared to other schemes [17–21]. In this study, a recent approach [17] for a parallel tridiagonal solver has been applied with the following features: the new solver, under the model of the SPIKE algorithm [20,21],

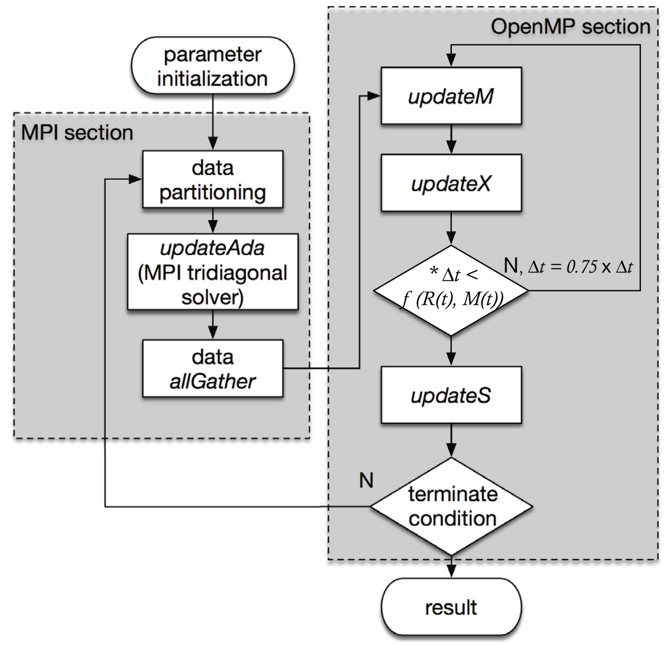

Fig. 2. Flow chart for hybrid MPI and OpenMP scheme. The equation to maintain the CFL condition is marked by $*$.

utilizes message passing interface (MPI) for interprocessor communication. In addition, the collective communication is not required and the machine-zero accuracy is guaranteed. The solver can compute a number of distinct tridiagonal systems together by being executed only once which is more efficient than being called multiple times to solve each single system. Multiple boundary conditions, such as periodic boundary conditions and natural boundary conditions, can be achieved without modifying the solver. Therefore, the solver is used in the function *updateAda*.

For the rest of the functions, both the shared-memory multiprocessing interface OpenMP and MPI have been implemented for parallelization. Thus, combined with the function *updateAda* under MPI, the whole algorithm is parallelized using two approaches: hybrid MPI combined with OpenMP, and pure MPI.

2.2.1. Hybrid MPI combined with OpenMP approach

In the hybrid scheme, the MPI section and the OpenMP section are designed to be independent of each other to reduce the implementation difficulty. A flow chart is shown in Fig. 2 to represent the architecture. At first, the input data is partitioned into multiple subsets, each of which is assigned to a particular node, to satisfy the precondition of the MPI tridiagonal solver in the *updateAda* function. Fig. 3 shows an example of the spatial decomposition for the case with a 5-sided polygonal shock in which the shock front has been split into 21 points that are distributed among five processors. It can be seen that the first point and the last point on the shock front share the same location because of the closed geometry. After calculation by the MPI tridiagonal solver, the resulting data is gathered into one node as a whole, and then, broadcasted to all nodes. The purpose of this broadcast is to guarantee that every node has the entire dataset. The gather and broadcast are referred as *allGather* in MPI section. In the OpenMP section, each node conducts the same computation by using its own processors with the same spacial decomposition method as shown in Fig. 3. At the end of each iteration, each node will contain the whole resulting data. Consequently, there is no cost due to communication between nodes in the data partitioning step, so in other words, the communication only happens in the MPI tridiagonal solver and the *allGather* steps. The main advantage

Table 1
Computational cost of each function for three different tests. A^* and N^* denote the apothem and the total number of points of the polygon.

	$A^* = 1\, N^* = 3{,}242$		$A^* = 10\, N^* = 32{,}491$		$A^* = 100\, N^* = 324{,}911$	
	Time (s)	Percentage	Time (s)	Percentage	Time (s)	Percentage
updateAda	21.5	64.56%	223.19	65.01%	2539.98	67.82%
updateM	9.36	28.11%	94.52	27.53%	937.56	25.03%
updateX	1.13	3.39%	11.99	3.49%	115.32	3.08%
updateS	0.75	2.25%	8.38	2.44%	92.01	2.46%
Time-step reduction	0.11	0.33%	1.05	0.3%	10.78	0.29%
Smooth function	0.188	0.56%	2.12	0.60%	20.51	0.55%

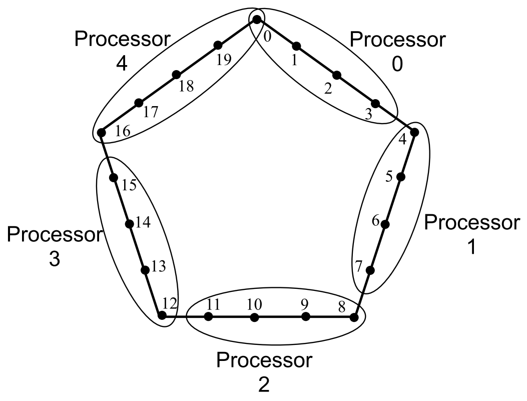

Fig. 3. Example of spatial decomposition of a 5-sided polygonal shock front split into 21 points distributed among five processors.

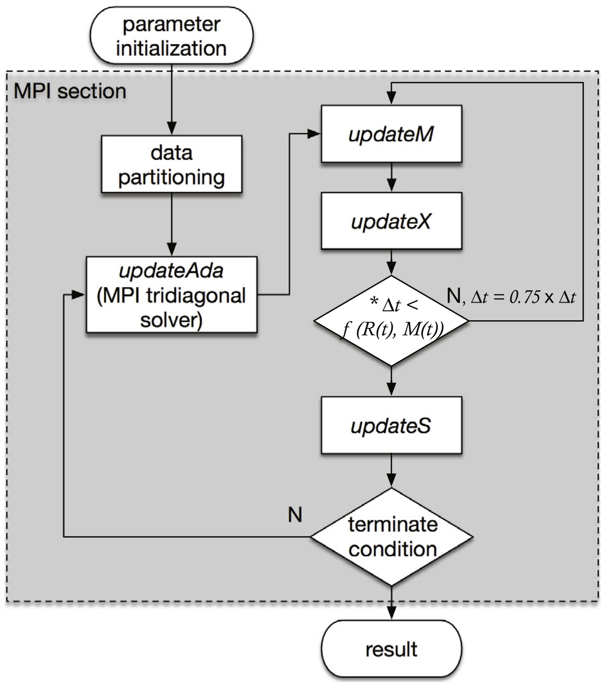

Fig. 4. Flow chart for MPI scheme. The equation to maintain the CFL condition is marked by $*$.

of this approach is that all processors share the same memory, and as a result, parallelization can be easily achieved for all the loop-independent functions with little communication cost. The implementation process is considerably simpler compared to the MPI approach because of the limited amount of data communication.

2.2.2. MPI approach

The pure MPI structure is presented in Fig. 4. In the beginning, all the data is partitioned and assigned into multiple nodes. Compared to the previous hybrid approach, the data-gathering step is eliminated by rescheduling data allocation for loop-carried functions so that each node can conduct its own work without requiring additional information from the other nodes. To be specific, in the functions *updateAda* and *updateS*, point-to-point communications have been used to allocate the data to avoid data dependency across adjacent nodes. Although this approach increases the difficulty in both the implementation and debugging process, it is expected to achieve a better parallel performance than the previous approach.

2.3. Comparison with analytical solution

To compare the two parallel schemes, numerical simulations of regular polygonal converging shocks with 9 sides (enneagon) and 10 sides (decagon) have been conducted on the above schemes with the same initial conditions as indicated in Section 2.2 and the apothem is one. Fig. 5 shows the numerical results, in which the successive shock fronts of each iteration sequence are plotted. As the polygonal converging shock propagates towards the center, the original shape keeps reshaping and a Mach stem is generated at each corner. In Fig. 5, r_0 and M_0 represent the distance from the center to the initial side of the polygon and the initial Mach number, respectively, while r_1 and M_1 denote the distance from the center to the first repeated polygon of the same shape as the initial polygon, and its Mach number. From Table 2, it can be seen that the numerical results from the parallel schemes agree well with the analytical solutions.

3. Results and discussion

3.1. Benchmark

Performance analysis has been conducted for the two parallel schemes on the high performance computing facility (HPCC) at University of Southern California. The nodeset, HPCC sl250s, consists of 256 Dual-Octacore Intel Xeon CPUs operating at 2.4 GHz with 64 GB memory. A strong-scaling test has been performed for both schemes by simulating a decagon converging shock with a total number of 3242,911 discrete points. The initial conditions are the same as those described in Section 2.2. As previously mentioned, in the hybrid scheme the MPI section and the OpenMP section are independent of each other. To perform the strong-scaling test for both sections, the number of nodes is defined as N_{node} and the number of processors per node is denoted as P_{pn}. In addition, the granularity in the MPI section can be defined as N/N_{node}, where N denotes the number of discrete points on the shock front, while in the OpenMP section, the granularity is defined as N/P_{pn}. Then, the strong-scaling test for the hybrid scheme is performed as follows: the granularity in both sections is kept consistent with each other, which means N_{node} is always set to be equal to P_{pn}, while N is kept constant. In this test, we define $\beta = N_{node} = P_{pn}$, which ranges from 1 to 16. Consequently, the efficiency of the hybrid scheme is defined as the ratio of the speedup of the parallel code divided by β. Fig. 6 shows the wall-clock time and the efficiency for the hybrid scheme. As a baseline, the ideal case is calculated by the ratio of the

Table 2
Comparison of numerical results with analytical solutions from [16].

	Enneagon simulation (MPI)			Decagon simulation (hybrid)		
	Analytical	Numerical	Difference	Analytical	Numerical	Difference
M_1/M_0	1.175	1.161	1.2%	1.155	1.159	0.3%
r_1/r_0	0.442	0.446	0.8%	0.482	0.486	0.8%

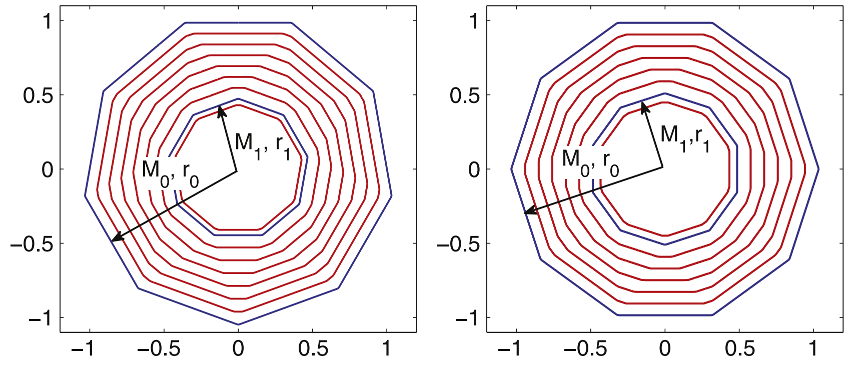

Fig. 5. Converging shock propagation showing successive positions for an enneagon (MPI scheme with 4 cores) and a decagon (hybrid scheme with 2 nodes and 2 processors per node). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

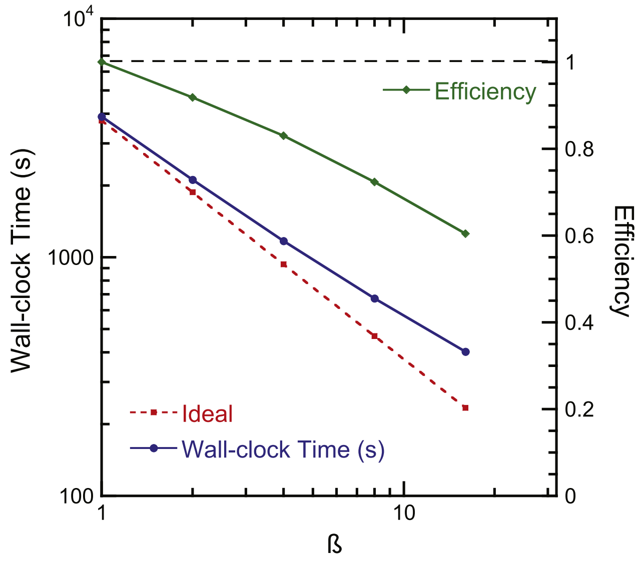

Fig. 6. Wall-clock time of strong scaling for hybrid scheme using 3242,911-point shock front on β nodes and β processors per node.

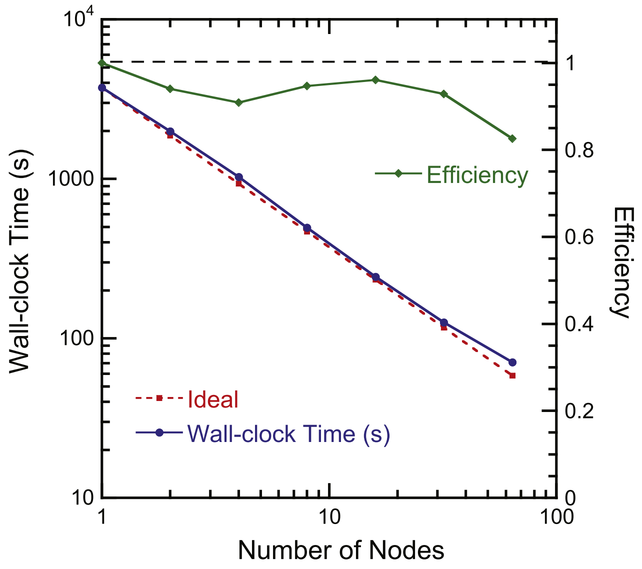

Fig. 7. Wall-clock time of strong scaling for MPI scheme using 3242,911-point shock front on N_{node} nodes.

wall-clock time for the case with $N_{node} = P_{pn} = 1$ over the number of different β.

For the MPI scheme, the strong-scaling test is performed with the same setup as above, except that N_{node} is varied from 1 to 64 and P_{ppn} is set to 1 for all cases. Therefore, the efficiency of the MPI scheme is defined as the ratio of the speedup of the parallel code divided by N_{node}, and the ideal case is calculated by the ratio of the wall-clock time on a single node over the number of different nodes. As shown in Fig. 7, the hybrid scheme has a comparable performance with the MPI scheme up until the case of $\beta = 8$. The efficiency of the hybrid scheme for the case of $\beta = 2$ is 0.92 and for $\beta = 4$ it is 0.83. However, as β increases, the efficiency drops from 0.72 with $\beta = 8$ to 0.6 with $\beta = 16$ cores. In contrast, the efficiency of the MPI scheme stays above 0.9 until 64 nodes are used. The main reason of the difference between the two schemes is that there is a data gather-broadcast step in the hybrid scheme to connect the MPI section with the OpenMP section, which is denoted as *allGather* in Fig. 2. This additional, albeit necessary step, makes the communication/computation ratio in the hybrid scheme higher than that of MPI scheme. With the increment of N_{node} and P_{pn} in use, the communication/computation ratio increases, and results in a loss of performance. In the MPI scheme, without this extra *allGather* step, the wall-clock time decreases nearly linearly with the granularity (N/N_{node}). The reason that the efficiency in the MPI scheme drops slightly for the case with $N_{node} = 2$ is likely due to that the communication cost is introduced in the computation compared to the case with a single node. When the number of nodes increases to 64, the communication cost starts to have a notable effect on the performance.

Additionally, an isogranular-scaling test, where the number of discrete points on the shock front in each node, N/N_{node}, is fixed, has been performed for the MPI scheme. The test case is a decagonal shock with $25{,}991N_{node}$ discrete points while N_{node} ranges from 2 to 128 and P_{pn} is set to be 1 for all cases. The weak scaling efficiency for the case of N_{node} nodes is calculated as t_2/t_n, where t_2 and t_n denote the wall-clock time for a 2-node test and a N_{node}-node test, respectively. Fig. 8 shows the wall-clock time versus the number of nodes scaled with the number of points. The

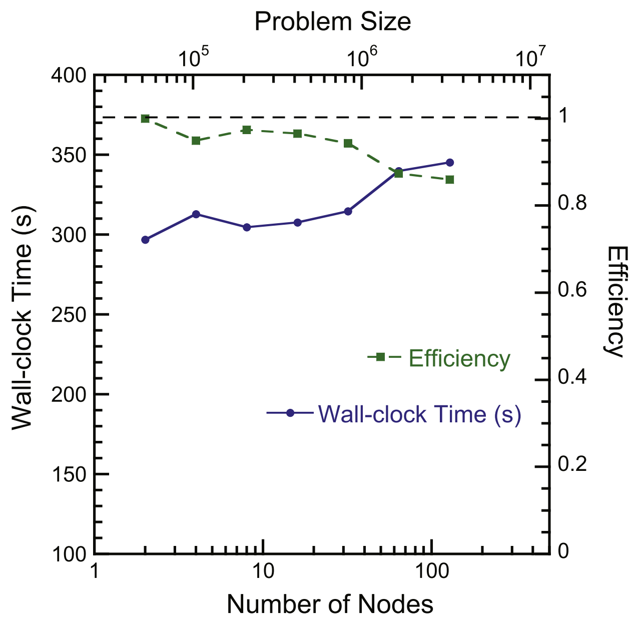

Fig. 8. Wall-clock time of scaled workloads using 25,991N_{node}-point shock front on N_{node} nodes.

efficiency remains nearly constant around 0.95 until the case of 64 nodes, which indicates a good scalability.

3.2. Symmetric boundary condition

A symmetric boundary condition has been developed to further improve the speedup for converging shock configurations. From Fig. 5, it can be observed that as the polygonal converging shock propagates towards the center, the lines of symmetry of the shock front is independent from its reconfiguration process. Thus, it is possible to compute only one section of the shock front instead of the full geometry. This symmetry feature indicates that the norm vector at each point where the line of symmetry meets with the shock front is always directed to the center. This is the boundary condition for each symmetric part of the shock front. For example, since the shock front features a polygonal shape, geometrical symmetry is used such that only a triangular portion of the converging shock, from a corner where two shocks meet to the mid-point of a neighboring planar side, is considered. The shock velocity on either side of this triangle is always directed along the sides of the triangle, i.e. the symmetry lines. With this condition, the problem size can be reduced from N down to $N/2n_l$, where N denotes the number of discrete points and n_l denotes the number of symmetric lines. A test on 12-sided polygonal (dodecagon) converging shock has been performed to verify the boundary condition. Three different symmetric parts (1/2, 1/4 and 1/24) have been computed independently and compared with the full size, see Fig. 9. It can be seen that all of the symmetric parts match the full size result very well. For performance analysis, the speedup of the symmetric boundary condition is defined as wall-clock time ratio of the full geometry versus the reduced geometry using a single core. Results are summarized in Table 3, and it is shown that for the dodecagon shock front, the speedup can be obtained up to 19.26.

Table 3
Computational cost and speedup of 1/2, 1/4, 1/24 of the full geometrical simulation using symmetric boundary conditions. All the simulations are terminated after 10,000 iterations.

Problem size	Full size	1/2 size	1/4 size	1/24 size
Computational cost	17.15	8.65	4.44	0.89
Speedup	1.0	1.98	3.86	19.26

4. Conclusions

In this study, a spatial decomposition method to implement the GSD simulation on parallel computers was adopted. At first, performance profiling was conducted to analyze computational hot-spot in the serial scheme. Two schemes were designed to parallelize the simulation, in which MPI and OpenMP were employed. An efficient tridiagonal solver [17] based on the SPIKE algorithm [20,21] was also incorporated into the parallel implementation to handle the most computationally expensive function.

Performance analysis and comparison between the two schemes were investigated. The hybrid scheme shows its advantage of implementation ease and the running result demonstrates its speedup, 9.7, on 16 HPCC node-processors. On the other hand, the MPI scheme leads to an improved performance in efficiency and scalability. The strong-scaling experiment proves a high efficiency of 0.93 using 32 HPCC cores, while the isogranular-scaling experiment shows that the efficiency holds up to 0.83 using 64 HPCC cores.

To further improve the speedup for symmetric converging shock simulations, symmetric boundary conditions were developed to reduce the problem size considerably. Results have shown that for a dodecagonal converging shock front, a speedup of up to 19.26 can be achieved.

Although this study only focuses on converging shock configurations, the parallel schemes can be easily extended to solve more generalized shock setups by changing the boundary conditions. In the future, parallelization on three dimensional GSD model will be investigated.

References

[1] A.G. Mulley Jr., N. Engl. J. Med. 314 (13) (1986) 845–847.
[2] C.K. Li, A.B. Zylstra, J.A. Frenje, F. Séguin, N. Sinenian, R.D. Petrasso, P.A. Amendt, R. Bionta, S. Friedrich, G.W. Collins, et al., New J. Phys. 15 (2) (2013) 025040.

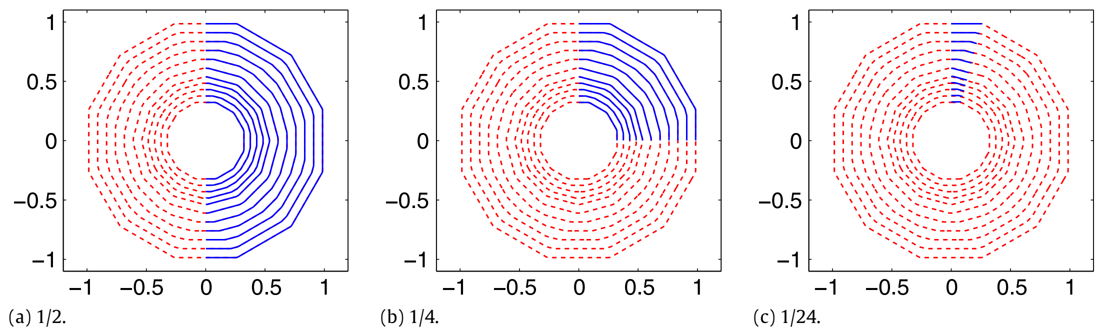

Fig. 9. Dodecagon shock front propagation at successive time instants. Red dashed line represents the full geometrical simulation, blue solid line represents, from left to right, 1/2, 1/4, 1/24 of the full size simulation using symmetric boundary conditions.

[3] G. Iosilevskii, D. Weihs, J. R. Soc. Interface 5 (20) (2008) 329–338.
[4] G. Guderley, Luftfahrtforschung 19 (9) (1942) 302–312.
[5] G.B. Whitham, J. Fluid Mech. 2 (02) (1957) 145–171.
[6] G.B. Whitham, Linear and Nonlinear Waves, Vol. 42, John Wiley & Sons, 2011.
[7] A.E. Bryson, R.W.F. Gross, J. Fluid Mech. 10 (01) (1961) 1–16.
[8] W.D. Henshaw, N.F. Smyth, D.W. Schwendeman, J. Fluid Mech. 171 (1986) 519–545.
[9] D.W. Schwendeman, Proc. R. Soc. Lond. A Math. Phys. Eng. Sci. 416 (1850) (1988) 179–198.
[10] D.W. Schwendeman, Proc. R. Soc. Lond. A Math. Phys. Eng. Sci. 441 (1912) (1993) 331–341.
[11] Y. Noumir, A. Le Guilcher, N. Lardjane, R. Monneau, A. Sarrazin, J. Comput. Phys. 284 (2015) 206–229.
[12] D.W. Schwendeman, J. Fluid Mech. 188 (1988) 383–410.
[13] J.P. Best, Shock Waves 1 (4) (1991) 251–273.
[14] J.P. Best, Shock Waves 2 (2) (1992) 125–125.
[15] N. Apazidis, M.B. Lesser, J. Fluid Mech. 309 (1996) 301–319.
[16] D.W. Schwendeman, G.B. Whitham, Proc. R. Soc. Lond. A Math. Phys. Eng. Sci. 413 (1845) (1987) 297–311.
[17] D. Ghosh, E.M. Constantinescu, J. Brown, SIAM J. Sci. Comput. 37 (3) (2015) 354–383.
[18] H.S. Stone, J. ACM 20 (1) (1973) 27–38.
[19] H.H. Wang, ACM Trans. Math. Software 7 (2) (1981) 170–183.
[20] E. Polizzi, A.H. Sameh, Parallel Comput. 32 (2) (2006) 177–194.
[21] E. Polizzi, A. Sameh, Comput. Fluids 36 (1) (2007) 113–120.