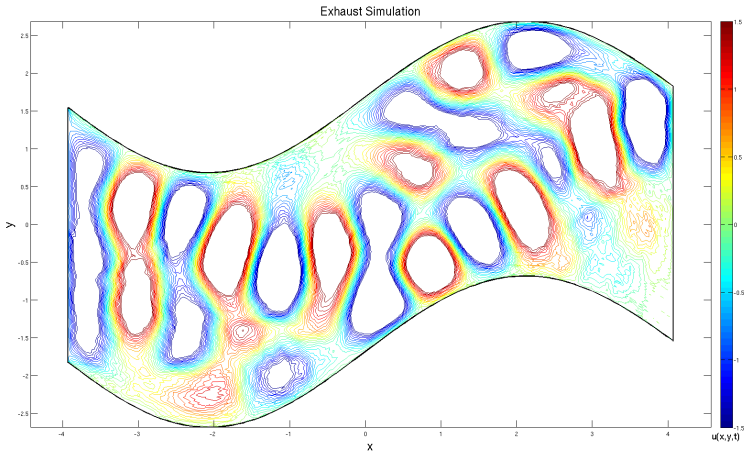
Solving the Wave Equation On a Curvilinear Grid Using MPI CS/MATH 471

Teo Brandt and Brennan Collins

December 10, 2015

Abstract



In this report the hyperbolic equation known as the 2D wave equation is presented with a constant coefficient

$$u_{tt} = c(u_{xx} + u_{yy}) + F(x, y, t),\ (x, y) \in \Omega, t\ 0,$$

with Dirichlet boundary conditions

$$u(x, y, t) = g(x, y, t),\ \forall (x, y) \in \delta\Omega,$$

and initial conditions

$$u(x, y, 0) = f(x, y),\ u_t(x, y, 0) = h(x, y), \forall (x, y) \in \Omega$$

1 Objective

In this project we will restrict the geometry Ω to be a logically square shaped domain, assuming that there is a smooth mapping $(x, y) = (x(r, s), y(r, s))$ from the reference element $\Omega_R = \left\{(r, s) \in [1, 1]^{\,2}$ to Ω.
The domains that will be considered in this report are shown below.

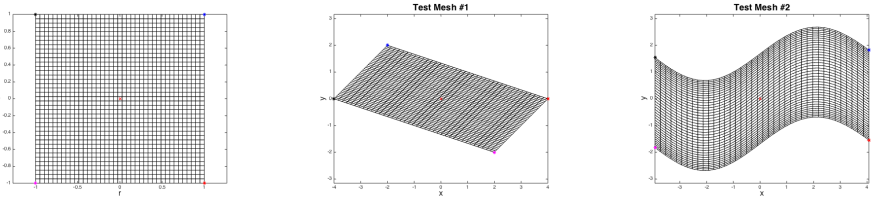


Figure 1: Spacial Domains 1-3

1.1 Details

In the (r, s) coordinate system the wave equation takes the form: (1)

$$Ju_{tt} = c((Jr_x(r_x u_r + s_x u_s) + Jr_y(r_y u_r + s_y u_s))_r$$

$$+(Js_x(r_x u_r + s_x u_s) + Js_y(r_y u_r + s_y u_s))_s)$$

$$+JF(x, y, t),\ (x, y) \in \Omega, t\,0$$

where the metrics and jacobian are

$$s_x = \frac{-y_r}{x_r y_s - x_s y_r}; s_y = \frac{x_r}{x_r y_s - x_s y_r}; r_x = \frac{y_s}{x_r y_s - x_s y_r}; r_y = \frac{-x_s}{x_r y_s - x_s y_r}$$

and

$$J(r, s) = x_r y_s - x_s y_r$$

respectively.

2 Parallelization Assignments

The first step in parallelizing the problem is to decompose the domain in a way that minimizes communication between processors. The best choice in this case is to take a 2-dimensional decomposition. The decomposition that works best here for an MxN domain with P processors is to break the domain into $P = X_m * Y_m$ sections, where $\left|\frac{M}{X_m} - \frac{N}{Y_m}\right|$ is minimized.

Once $\left|\frac{M}{X_m} - \frac{N}{Y_m}\right|$ is minimized, we find px and py for each processor. Each processor is assigned an ID, which is a value from 0 to P-1. Using this ID along with the X_m and Y_m found above, we can assign px and py as follows:

$$px = \frac{ID}{X_m} + 1$$

$$py = IDmodY_m + 1$$

This assigns a unique (px,py) pair to each processor. We can also use the ID to determine our neighbors. Our north neighbor then would simply be ID - 1, the south neighbor would be ID + 1, the west neighbor will be $ID - Y_m$, and the east neighbor will be $ID + Y_m$.
Now that neighbors are assigned, we can perform communication effectively, similar to how it was done in the 1-D domain decomposition. The main difference is that communication is not just between left and right neighbors, but also top and bottom.
To ensure that communication was proceeding as expected, a test was added where each node sent its ID to its direct neighbors (N,S,E,W) and then write the updated to a file. Upon inspection of the file, the edges of array corresponded to IDs of the node's immediate neighbors, so 2-D communication is properly implemented.

3 Discretization in Time and Space

For discretization in time, we use a simple centered difference formula

$$u_{tt}(x_i, y_j, t_n) \approx \frac{v_{i,j}^{n+1} - 2v_{i,j}^{n} + v_{i,j}^{n-1}}{(\Delta t)^2}$$

One of the problems that occurred in the discretization of time for each processor, is that each node has its own matrices independent of the other processors. Since this is the case, each node will have its own unique Jacobian. Previously, the timestep in each processor was set right before the discretization loop, each processor was using its own Jacobian to set this timestep. Since the Jacobian could be unique for each node, then the size of the timestep and transitively the number of timesteps in the discretization loop for time could be different from processor to processor. That means that some of the processors will get through this loop before others. The processors that re-enter the loop will go on to make a *mpi_sendrecv* call to a neighbor that has completely passed through the loop, meaning that the call will go unanswered, and the node will enter deadlock. The solution was simple, to remove the Jacobian from the setting of the timestep and replace it with a numerical constant.

4 Manufactured Solution

The manufactured solution is tool that allows the algorithm to be tested against a known solution v to the numerical approximation u that we wish to achieve. In this case

$$u_{tt} = c(u_{xx} + u_{yy}) + F(x, y, t),\ (x, y) \in \Omega, t\ 0,$$
$$u(x, y, t) = g(x, y, t),\ \forall (x, y) \in \delta\Omega,$$
$$u(x, y, 0) = f(x, y),\ u_t(x, y, 0) = h(x, y), \forall (x, y) \in \Omega$$

By measuring error

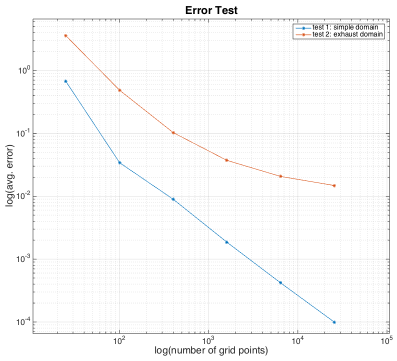
$$\epsilon_p(t) = \left(\int_0^1 \left| u - v \right|^p dx\right)^{\frac{1}{p}}$$

for some different values of h we can verify that order of our approximation is $\epsilon_p \sim O(h^r)$ as we expect. The results of testing the trigonometric manufactures solution of the form

$$u(x, y, t) = sin(t - x) * sin(y)$$

for the domains referenced in fig.1 are given In the figure it is important to

Figure 2: MMS convergence



note that the error is plotted against the number of grid points, which in this experiment was square $nx = ny$ so the order 1 shown is actually order 2.

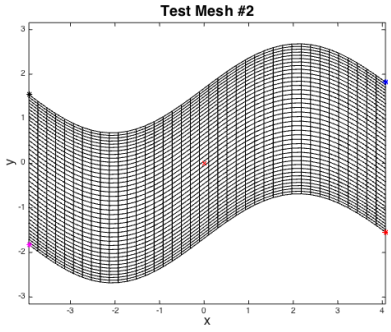
5 The Simulation of the Exhaust

The theory up to this point has been in preparation for a discussion and presentation of our approximation of the complicated curvilinear coordinate system shown below again for convenience with the smooth mapping

$$x(r, s) = 4r + 2sin(r) * 0.05sin(r)$$

$$y(r, s) = sin(3r) + 2sin(s)$$

Figure 3: Exhaust Grid



For this system boundary conditions were set as follows

$$u(0, y, 0) = 2sin(5t)$$

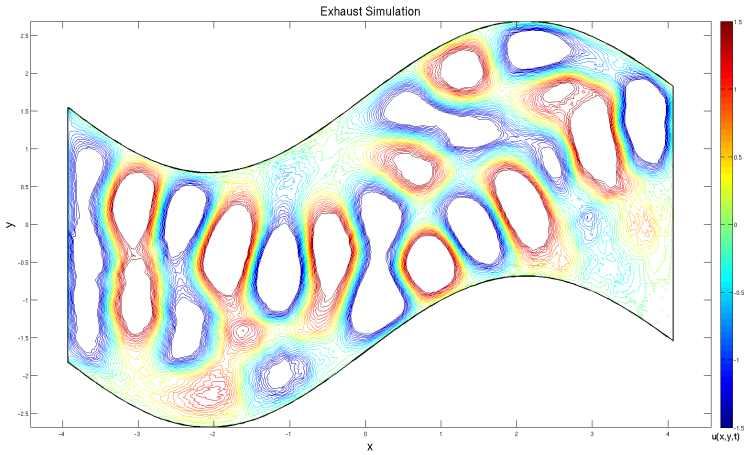
$$u(right, y, 0) = 0$$

$$u(x, bottom, 0) = 0$$

$$u(x, top, 0) = 0$$

The simulation was run for 10 "seconds" or $\sim 700 timesteps$.

Figure 4: Exhaust Simulation Results



6 Strong vs. Weak Scaling (Parallelization)

Before presenting results, it should be noted what strong and weak scaling are, and why they are different. For strong scaling, the size of the domain is fixed as the number of processors on the domain increase whereas for weak scaling, the size of the domain grows proportional to the number of processors as this number gets larger. The purpose of growing the domain proportional to the number of processors is to fix the domain size inside any of the processors.
For strong scaling we expect that as the number of processors increases, the cost of communication will also increase and thus will incur more overhead as a result. Then we expect that as P (number of processors) increases, the time it takes for the program to complete will approach a constant value. For weak scaling, since the domain is a fixed size inside each node for any number of processors, then the time it takes for the program to run to completion should remain constant. So then it is expected that weak scaling will be linear.

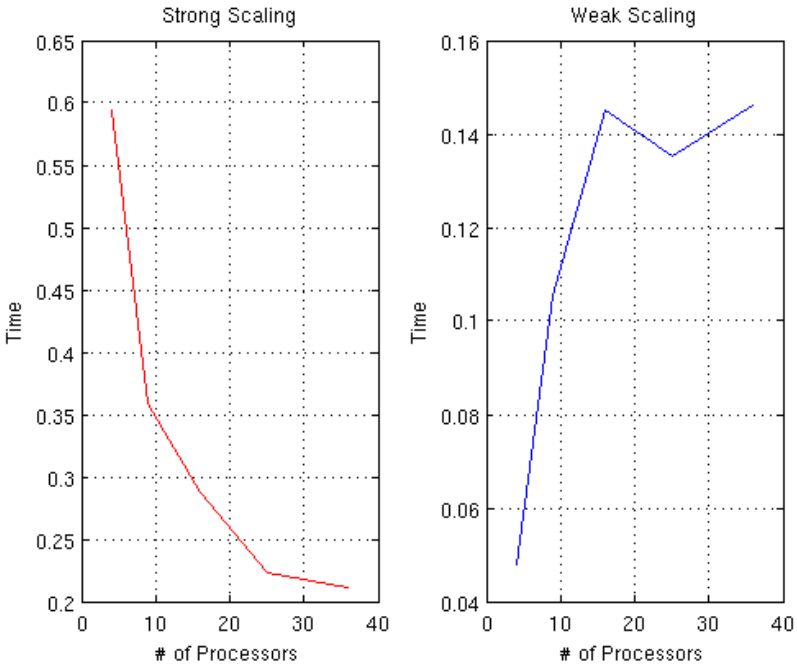
Table 1: Experimental results for Strong Scaling

Some	actual	content
n = 4	D = 100x100	time = 0.594162
n = 9	D = 100x100	time = 0.358611
n = 16	D = 100x100	time = 0.287129
n = 25	D = 100x100	time = 0.222724
n = 36	D = 100x100	time = 0.210599

Table 2: Experimental results for Weak Scaling

Some	actual	content
n = 4	D = 20x20	time = 0.047831
n = 9	D = 30x30	time = 0.105451
n = 16	D = 40x40	time = 0.145203
n = 25	D = 50x50	time = 0.135175
n = 36	D = 60x60	time = 0.145983

Figure 5: Strong vs. Weak Scaling



As you can see from the results, our predictions held true for both strong and weak scaling. For strong scaling, it is clear that increasing the number of processors becomes less and less effective, and for weak scaling, the time it takes to complete just about remains constant, which fits the definition of linear scaling for a weak scaling program.

7 Appendix

In order to compile and execute the code for this assignment a make file is used:

/Homework/Homework7/Code/

Once in this directory the following command will compile and execute the code:

$ make && ./homework7.x

Included in the same directory is a sample simulation video named *exhaustsim.avi*
MUST WATCH!

References

[1] Daniel Appelo *Homework 7*. referenced Dec. 10, 2015