Parallelization of a High-Order Accurate Unstructured Mesh Finite-Volume Solver

K. Michalak and C.F. Ollivier-Gooch

Department of Mechanical Engineering, University of British Columbia, Vancouver, BC, V6T 1Z4, Canada

Email: *michalak@mech.ubc.ca*

ABSTRACT

A generic solver eliminates the need to write new finite-volume codes for each type of physics. By separating the physics from the numerics of the solver, a modular design is achieved. New physics modules can easily be written with a minimal knowledge of the finite-volume method. A parallel solver allows simulation of complex physics on intricate domains in a timely manner by using numerous processors simultaneously. In this paper we describe the steps needed to adapt a high-order accurate unstructured mesh generic finite-volume solver to a parallel architecture. A message-passing approach is used which allows the solver to operate on a distributed memory system, such as a cluster of workstations. The reconstruction stencil is determined at the preprocessing stage and an appropriate parallel data structure for the solution is formed. Fluxes for faces on the partition boundary are evaluated by communicating the reconstruction coefficients to the adjacent processor. Good performance scalability is achieved for second and fourth-order accurate solutions on cell and vertex centered meshes.

1 INTRODUCTION

Generic solvers have the potential to dramatically reduce the development time needed to solve a wide range of problems. By separating the numerics common to all finite-volume problems from physics specific portions of code, Ollivier-Gooch has developed the ANSLib library which provides a flexible framework for simulation [1]. However, the simulation time for complex problems on such a solver is far too large. This simulation time could be greatly reduced by adapting the solver to run on a parallel computer architecture. This paper outlines the changes made to a generic finite-volume solver to enable it to run in a scalable fashion on a distributed memory parallel computer. Parallel data structures and communication are handled using PETSc [2] and MPI [3] libraries.

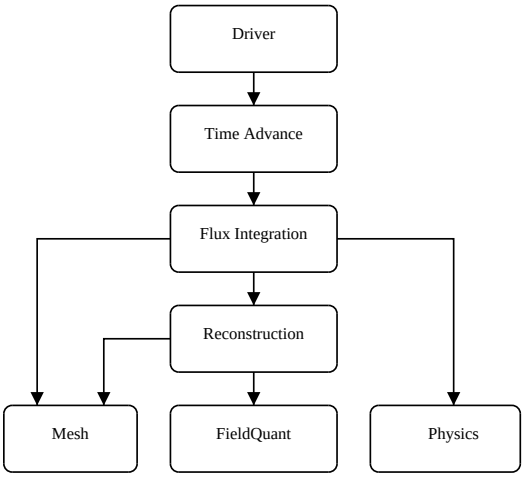

Figure 1: Simplified class interaction in ANSLib

2 ANATOMY OF A GENERIC SOLVER

The parallelization algorithm is tightly integrated with the other parts of the code, therefore an understanding of the structure of the generic solver is critical. The design and implementation of a generic solver lends itself well to Object Oriented Programming (OOP). We have chosen to use the C++ language, which provides good performance and third party library support. OOP allows a modular design that can be used to isolate the physics from the numerics of a problem. The end-user only needs to implement a few functions in the *Physics* class that describe the fluxes and boundary conditions of the problem. The object oriented design also facilitates handling a variety of mesh types. ANSLib currently works with both structured and unstructured (cell and vertex centered) meshes in 2D and 3D.

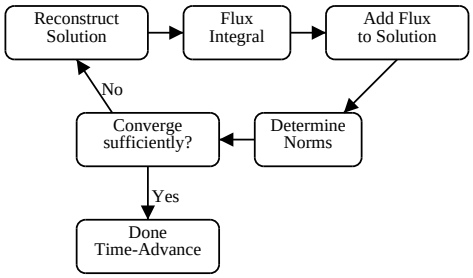

Figure 2: Multistage time advance to steady-state solution

A flow chart showing a simplified model of class interactions within ANSLib is shown in Figure 1. The *Driver* is responsible for parsing command-line parameters and creating the requested type of mesh and time advance objects. After these preprocessing steps are taken, the *Driver* calls a function in the time advance class. Although an implicit time-advance scheme is available [5], at this point in time only explicit multistage time advance has been parallelized. An outline of multistage time advance and its dependence on other classes in shown in Figure 2.

During reconstruction a piecewise polynomial representation of the solution within each control volume is found. This is done using a constrained least-squares problem, where the constraint is matching the calculated solution average in the control volume being reconstructed. The polynomial is made to minimize the error in predicting the solution in nearby control volumes. The details of k-exact reconstruction can be found in [6]. The coefficients of the reconstructed polynomial are stored within the *Reconstruction* object. Higher order accuracy can be achieved by using a higher order polynomial and increasing the number of neighbour control volumes used in the solution of the least squares problem.

The flux is found at each Gauss point on every face of the mesh. There are two reconstructed solutions for each Gauss point, one from each of the adjacent control volumes. Both of these values are passed to the flux function of the *Physics* class. The function can choose to use one (for example, if an upstream value is desired in an advection problem) or both (for problems with a diffusive component) of these solution values to determine the flux. The flux integration class takes care of calling the flux function for all of the Gauss points and adding the resulting fluxes to both adjacent control volumes.

ANSLib uses wave speed estimates provided by the flux function and a run-time provided CFL number to determine the maximum allowable time-step for each control volume. For steady-state problems local time-stepping can be used until the solution is sufficiently converged. Convergence is determined by the L_2 norm of the flux integral. For time-accurate solutions, the minimum stable local time-step is used globally for all control volumes until the desired point in simulation time is reached.

3 Parallel Algorithm Description

3.1 Mesh Partitioning

As is usual in parallel processing, the mesh is partitioned such that each processor is responsible for a distinct portion of the domain; for this we use ParMETIS[4]. A graph representing the connectivity of the control volumes needs to be supplied to the partitioning library. However, to do this in a memory scalable fashion, careful consideration must be made of how the mesh data is read from disk storage. Mesh data is read one chunk at a time by the first processor, sent to another processor, and deleted from the first processor before another chunk is read. To facilitate post-partitioning data distribution, each processor stores three different types of data: vertex location, cell connectivity, and face connectivity. Since the cells are not numbered in any particular order, the data stored on a single processor does not normally represent a continuous region. Figure 3 shows the distribution of cell connectivity information among processors before partitioning.

ParMETIS partitions the mesh such that each processor has approximately the same number of control volumes (to balance the load) and the number of graph links cut by the partition is minimized (to minimize communication). Using the results of partitioning, the control volume connectivity data is scattered to the processor it has been assigned to. Based on this information, data for the faces and vertices adjacent to the local cells can then be gathered from the other processors. An example of partitioning a cell centered mesh using ParMETIS is shown in Figure 4.

At the end of the mesh pre-processing stage each processor has access to the following:

- Vertex coordinates for all local control volumes.
- Face and cell connectivity data for local control volumes

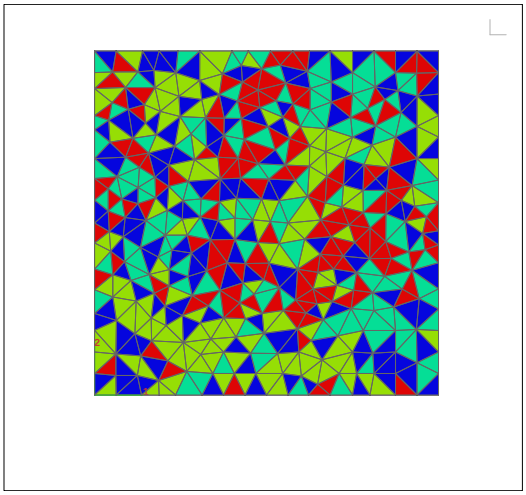

Figure 3: Distribution of cell connectivity data before partitioning

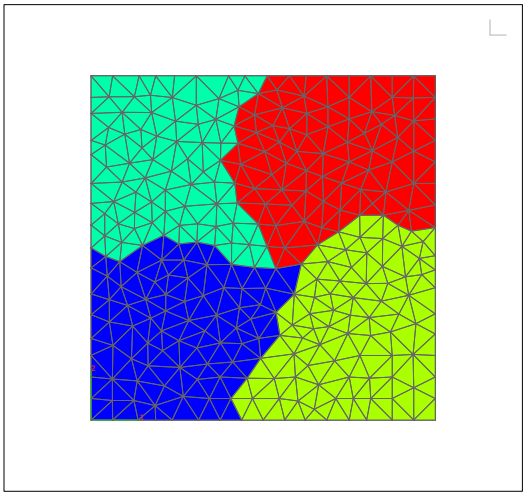

Figure 4: Example of mesh partitioning for 4 processors

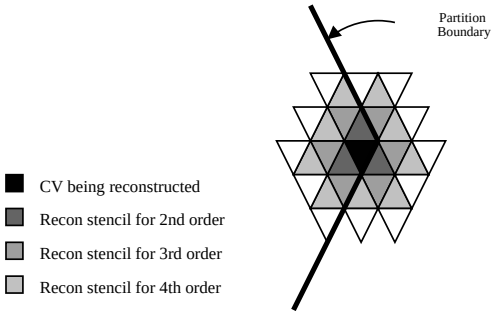

Figure 5: Example of a reconstruction stencil spanning across a partition boundary

- Reconstruction stencil for local control volumes
- Control volume reference location and moment data for all local control volumes and those that are in the reconstruction stencil of a local control volume.

As will be shown in the discussion of parallel reconstruction and flux integration, the numbering of control volumes and faces also needs to be carefully considered.

Currently, only unstructured 2D cell and vertex centered meshes have been implemented in parallel. However, the extension of these parallelization techniques to 3D unstructured and 2D/3D structured meshes should be straightforward.

3.2 Reconstruction

Reconstruction for the majority of the domain can be done as for the single processor case. However, the reconstruction for some control volumes near the partition boundary will require solution values which are stored on another processor. Figure 5 gives an example of the reconstruction stencil of a single control volume located near a partition boundary. Data retrieved from other processors is stored in special memory placeholders, called “ghost” values, which are allocated in the solution data structure (*FieldQuant*). These ghost values are filled by communicating with the other processors at each time-step. To simplify this and other parts of the solver, it is convenient to renumber local control volumes during mesh preprocessing such that they represent a continuous range starting from zero. Foreign control volumes whose solution values will be required during reconstruction are also assigned a new local numbering which begins after the last local control volume. Information on where solution data can

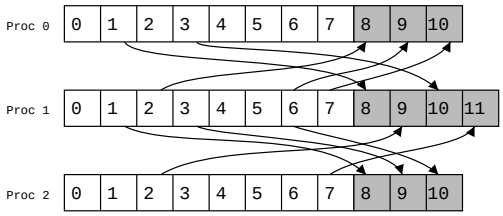

Figure 6: A parallel *FieldQuant* data structure

be found to properly fill these ghost values is also determined at preprocessing. An example of a parallel *FieldQuant* data structure can be seen in Figure 6.

To minimize the effects of communication latency on the performance of parallel reconstruction, communication can be overlapped with computation. Communication of the ghost values of the solution is started as soon as reconstruction begins. While the data is transfered, which can take several milliseconds on networks of workstations, reconstruction of the control volumes that are not near the partition boundary can take place. For this purpose, during preprocessing, a list of control volumes whose reconstruction stencil include only local control volumes is generated. Once reconstruction for these interior control volumes is completed, the program waits, if necessary, for the communication of the ghost *FieldQuant* values to finish. Reconstruction for the remaining control volumes can then take place.

3.3 Flux Integration

To evaluate the flux at a Gauss point on a face, reconstructed values of the solution from both adjacent control volumes are needed. Therefore we will need to give special consideration to faces which fall on the partition boundary. We again use the concept of ghosting to allocate storage for reconstruction coefficients of foreign control volumes. The ghosting stencil is not necessarily identical to that of the *FieldQuant* class. Only the reconstruction coefficients of the control volumes which fall immediately on the opposite side of a partition face are required. However, it is convenient to note that the ghosting stencil of the reconstruction coefficients will always be a subset of the *FieldQuant* ghosting stencil, except for first-order accurate flux evaluation when no *FieldQuant* ghosting is required. Therefore it is possible to number control volumes so that the ghost values of both *FieldQuant* and reconstruction coefficients represent a continuous range starting immediately after the last local control volume.

Although it would be possible to have one of the two adjacent processors evaluate the flux on a partition boundary face and communicate that value with the other processor, this would add an additional communication step. We therefore calculate the flux at the boundary face independently on each of the adjacent processors. Communication latency is hidden by calculating the fluxes at faces which do not fall on a partition boundary and evaluating the source term while the ghosted reconstruction coefficients are communicated.

3.4 Multistage Time Advance

For steady-state problems, the L_2 norm of the flux integral must be evaluated to determine convergence. Since each processor only knows about the flux values for its local control volumes, there needs to be some communication to determine if all processors have sufficiently converged their solutions. This is done by calculating a local L_2 norm and weight and communicating this value to a single processor. This processor adds the weighed L_2 norms together to determine an overall norm. This value is then broadcast back to all of the other processors. A number of options are available to reduce the latency associated with these two communication cycles. Since the multistage time advance scheme normally takes a large number of iterations to converge, we could calculate the L_2 norm only at a fraction of those time-steps. Alternatively, the L_2 norms could be communicated while the next time-step is undertaken; in this case one more time-step than is required for convergence would always be taken. Our current implementation of ANSLib does not use either of these strategies since, as will be shown in the results, the communication latency was not found to be a significant performance bottleneck.

4 Sample Problem Description

We have chosen to simulate the heat conduction problem to profile the performance of these parallel algorithms. The heat equation can be expressed as:

$$\frac{\partial T}{\partial t} - \frac{\partial^2 T}{\partial x^2} - \frac{\partial^2 T}{\partial y^2} = S(x,y)$$

The source term on the right-hand side must be specified as part of the problem statement. We can rewrite this equation in divergence form:

$$\frac{d\bar{T}_i}{dt} + \oint_i -\nabla T \cdot \hat{n} ds = \int_i S(x,y) dA$$

For our test case, we will consider a square domain $[0,1] \times [0,1]$ with homogeneous Dirichlet boundary conditions and the following source term:

$$S(x,y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$$

The exact steady-state solution to this problem is

$$T(x,y) = -\sin(\pi x) \sin(\pi y)$$

5 RESULTS

Since the parallel algorithm does not change the reconstruction stencil or flux evaluation, the results of the simulation can be verified to be identical to the original serial version of ANSLib. The accuracy of the serial version of ANSLib has been previously verified [1] , and it will not be repeated here.

A cluster of workstations networked with 100 Mbps Ethernet was used to run the simulation of the sample problem. To identify the performance bottlenecks, and to verify the validity of the algorithm, a number of parameters were varied:

- Mesh type
- Mesh size
- Order of accuracy
- Number of processors
- Communication overlap with computation

Each trial run was restricted to 100 iterations to allow profiling of a wide range of parameter combinations. The execution time used to determine speedup measured only the time-advance stage of the code. Convergence typically requires several thousand iterations and the execution time of mesh input and solution output stages becomes negligible for these real-world problems. The results presented use as a reference the run-time of the parallel code on one processor, which in our case is less than the run-time of the original serial code due to a few optimizations that took place during the parallelization. Parallel speedup is defined as:

$$S_P = \frac{t_1}{t_P}$$

where t_1 is the single processor run-time and t_p is the parallel run-time for P processors. The best speedup

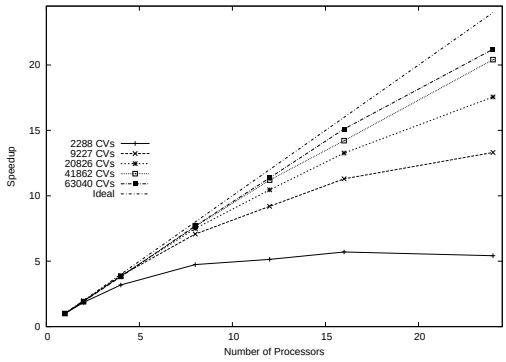

Figure 7: Speedup for second-order solution on a cell centered mesh without latency hiding

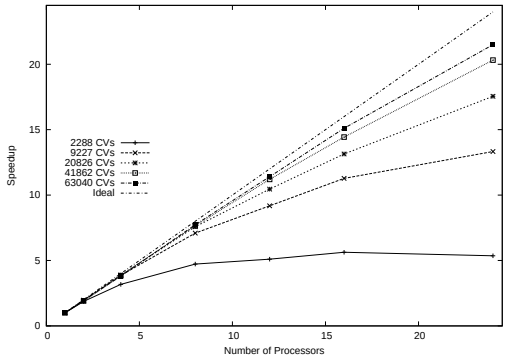

Figure 8: Speedup for second-order solution on a cell centered mesh with latency hiding

that can theoretically be expected is equal to the number of processors used. In a few cases we observe super-linear speedup which can be explained by the smaller data structures resulting from parallelization which increase the number of memory cache hits.

We begin by looking at the second-order accurate solution on an unstructured cell centered mesh in Figure 7. To evaluate the significance of communication latency we do not overlap the communication of solution data and reconstruction coefficients with computation for this run. We can see that the speedup approaches ideal values except for meshes with a small number of control volumes run on a large number of processors.

The experiment is repeated with communication/computation overlap enabled in Figure 8. No net improvement in performance can be seen in these results, which indicates that communication latency is not a major performance bottleneck.

The performance of a fourth-order accurate solution on an unstructured cell centered mesh is demonstrated in Figure 9. The increase in accuracy is achieved by us-

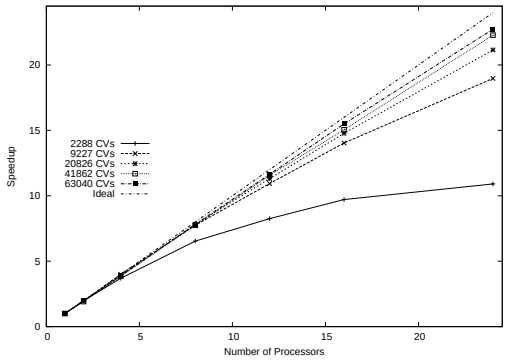

Figure 9: Speedup for fourth-order solution on a cell centered mesh

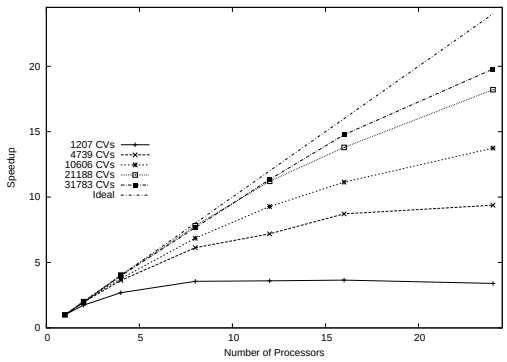

Figure 10: Speedup for second-order solution on a vertex centered mesh

ing two Gauss points per face for flux evaluation and by reconstructing the solution to a third-order polynomial. This in turn requires a larger reconstruction stencil and therefore increased amounts of "ghosting" of the solution data. However, the increased amount of computation required for reconstruction and flux evaluation effectively masks this extra communication.

Speedup for a second-order accurate solution on a vertex centered unstructured mesh is shown in Figure 10. These results indicate that the parallel method developed applies equally well to vertex centered problems.

The poor performance for problems on few control volumes run on a large number of processors can easily be explained by the relative processing power involved in communicating and copying data to the right locations. Specifically, the size of the ghosting stencil relative to the mesh size increases as the number of control volumes decrease.

To explain the performance limitation for larger meshes, we examine the results of partitioning in Table 1. Since the mesh partitioning does not assign exactly

Total CVs	Max CVs/proc	Expected speedup
2288	98	23.35
9227	395	23.36
20826	890	23.40
41862	1814	23.08
63040	2726	23.13

Table 1: Maximum speedup based on known load imbalance for 24 processors

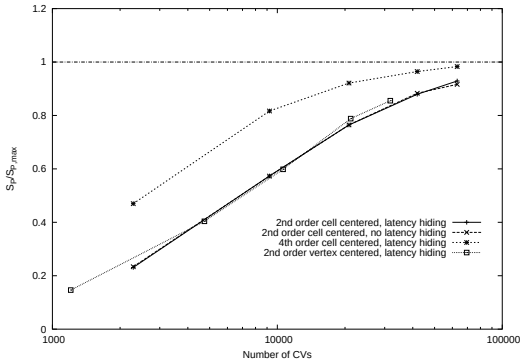

Figure 11: Speedup achieved on 24 processors relative to maximum expected speedup based on known load imbalance

the same number of control volumes per processor, certain load imbalances can be expected. Communication steps need to be taken at every time-step, which limits the speed of execution of the solver to that of the processor assigned to the heaviest load. Since the size of reconstruction stencils varies, the number of control volumes is only an estimate of the load. Nonetheless we can find an estimate of the maximum speedup by considering the processor with the largest number of control volumes. The maximum expected speedup is therefore:

$$S_P = \frac{CV_{max}}{CV_{tot}}$$

where CV_{max} represents the maximum number of control volumes assigned to a single processor and CV_{tot} represents the total number of control volumes in the mesh. The actual speedup for the 24 processor case is compared to the maximum expected speedup in Figure 11.

6 CONCLUSION

We have shown that the parallelization of a high-order accurate generic solver can be done efficiently by ghosting the solution data and the reconstruction

coefficients. Despite the increased size of the communication stencil for fourth-order accurate solutions, the speedup for a given mesh size was larger than for the second-order accurate solution. Ollivier-Gooch has previously shown that higher-order accurate methods have the potential to reduce computational effort required for a given level of solution accuracy [6]. The work presented herein demonstrates that this remains true when the algorithm is parallelized. At present, the performance of our algorithm on large meshes is primarily limited by the imbalance in the number of control volumes assigned to each processor by the mesh partitioner.

There remain a number of avenues to explore in the parallelization of the ANSLib library. Future work will include testing the algorithm on a larger number of processors to determine if other performance bottlenecks become important on larger parallel systems. We will also parallelize the implicit time advance scheme already implemented in ANSLib. The parallel solution of coupled multi-physics problems, which has already been implemented in the serial version of ANSLib [7], will also be considered in the future.

ACKNOWLEDGEMENTS

This work has been supported by the Canadian Natural Science and Engineering Council under grant OPG-0194467.

REFERENCES

[1] Carl F. Ollivier-Gooch. A Toolkit for Numerical Simulation of PDE's: I. Fundamentals of Generic Finite-Volume Simulation. Computer Methods in Applied Mechanics and Engineering, v 192 (9-10), pp 1147-1175, 2003.

[2] Satish Balay and Kris Buschelman and William D. Gropp and Dinesh Kaushik and Matt Knepley and Lois Curfman McInnes and Barry F. Smith and Hong Zhang. PETSc home page. http://www.mcs.anl.gov/petsc, 2003.

[3] William Gropp et al. MPI: The complete Reference, The MIT Press, Cambridge, Massachusetts, London, England, 1998.

[4] George Karypis, Kirk Schloegel and Vipin Kumar, PARMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, University of Minnesota, Dept. of Computer Sci. and Eng., 1998.

[5] Amir Nejat and Carl Ollivier-Gooch. A High-Order Accurate Unstructured GMRES Solver for Poisson's Equation. In Proceedings of the Eleventh Annual Conference of the Computational Fluid Dynamics Society of Canada, pp. 344-349, 2003.

[6] Carl F. Ollivier-Gooch and Michael Van Altena. *A* High-order Accurate Unstructured Mesh Finite-Volume Scheme for the Advection-Diffusion Equation. Journal of Computational Physics, v 181 (2), pp 729-752, 2002.

[7] Charles Boivin and Carl Ollivier-Gooch. A Toolkit for Numerical Simulation of PDE's: II. Generic Solution of Multiphysics Problems. Submitted to Computer Methods in Applied Mechanics and Engineering, April 2003.