Computation Techniques in a High Performance Parallel Simulation of Gas Dynamics in a Combustion Chamber

L. Környei

Széchenyi István University, H-9026, Győr, Egyetem tér 1.
Phone: +36-96-503-400/3150
e-mail: leslie@sze.hu

Abstract: The article describes several techniques utilized in the author's study of gas flow in a combustion chamber. Various models of Compressible Fluid Dynamics (CFD) are used in conjunction with the Finite Volume Method (FVM). Piston and valve movement is introduced with the grid snapping method.

Several methods are investigated and implemented in order to prepare the simulation software for high performance computing environments. Some of these methods (MPI, OpenMP, and their hybrid), and their feasibility is discussed below. It proves, that when starting the proper number of MPI processes, and the adequate number of threads for each process – these are determined by the hardware environment – the MPI-OpenMP hybrid method is scalable for the underlying problem for up to 200 processor cores, with a speedup of 60.

Two other delicate problems are also hand-picked, that rarely surface in static space simulations: vertex-matching in neighboring meshes, and the transformation of local physical values with interpolation during the change of space discretization, especially when swapping the mesh. Appropriate algorithms are presented for both. These need considerably smaller resources in memory and processor time, which make them applicable even runtime.

Keywords: CFD, FVM, moving boundary, MPI, OpenMP

1. Introduction

There is a particular type of physical phenomena, where the actual extents of the space investigated is changing, deforming, or moving in a predefined way [1]. Special care is required to describe the dynamics of gases, even with conventional CFD models, to handle piston and valve movement in finite volume discretization [2]. Apart from the physical and mathematical exercises, the computational tasks are also challenging. This article is hand-picked from the later.

During the simulations, the k-component Euler equations [3] are solved, where the local state is described by k components of partial density, ρ_k, three dimensions of

impulse density, ρv_X, and an energy density scalar, e. The governing equations are as follows:

$$\begin{matrix}\rho_k \\ \rho v_X \\ e\end{matrix} + \nabla_Y \begin{matrix}\rho_k v_Y \\ \rho_k v_X v_Y + \delta_{XY} p \\ v_Y\ e + p\end{matrix} = 0. \quad (1)$$

Boundary and initial conditions are additional, furthermore an equation of state is also needed to complete the model. In this work basic no-slip boundary conditions are applied on the chamber walls, and isobar conditions on the inlet and outlet ports. This way gas flow is induced by the piston movement solely. Time advancement is calculated with the first order explicit Euler method [3]. Space discretization is done using FVM. Several parts of the engine are split into subdomains of structured and unstructured meshes. The first order Euler method is based on a first-neighbor stencil.

Figure 1. Combustion chamber during the intake phase. The intake valve is at the bottom position. The freshly drawn gas is depicted as the dark range, showing where its relative density reaches 50%. The outlet valve is closed, and the corresponding cells are rendered inactive, thus are invisible here.

There are several methods to handle piston movement, where additional neighboring regions are present. The multi-component method decomposes the meshes in a way that only one part is moving with the piston and is deforming in a time-dependent fashion. Additional equations are solved for each cell and time step, to obey mass-conservation [4]. Another method is using an overlapping grid, one of which is always moving in front of the piston. This requires a constant interpolation of the local state values, although deals with degenerate cells [5]. The final method, which is used in this study, is called snapper. A special structured grid is generated, and the last line of vertices is moved with the piston, until the cell layer reaches its half height. Then this cell layer is set inactive, and the next one is stretched to one and half of its height. Local data is recalculated with interpolation only in these layers. This action is referred to as "snapping" [6].

It is undoubtedly useful to use simulations capable of calculating the gas dynamics in a high performance computing environment. In the first part of the article a hybrid approach of parallelization is presented within the simulation on topic. It is shown, that the proper mixture of MPI [7] and OpenMP [8] techniques is capable of vastly benefit from the hardware used.

The next chapter is dealing with the challenging task of fitting multiple structured and unstructured meshes together. As most unstructured meshes are made manually using HyperMESH [9], there is the inevitable scatter, introduced by human error. Two neighboring meshes, that seem to have a common surface, may have unnoticeable scatter in their vertex coordinates. This error introduces unphysical anomalies in the simulation even in short runs. As these scatters are much smaller than the grid size, a proper pairing algorithm, one of which is presented here, can eliminate the problem.

As interpolating data is one of the major issues in this simulation, the last part will address methods, that are fast enough to handle local data transformation conveniently. Data transformation is not only required by the snapper algorithm of the piston motion, but also the more complex geometry of the valves. They have a less constricted shape, which introduces several problems in discretization. A basic type interpolation algorithm is shown, that is fast enough to use runtime.

2. Methods of parallelization

Both methods discussed in this part are in fact programming standards, which have been known for a long time now: MPI dates 1994, and OpenMP 1997. A few years later there were successful trials on combining these, resulting in a “mixed” or “hybrid” method [11]. The basic idea comes from combining independent accelerator methods for shared memory and distributed memory system. As for inter-node communications MPI is used, along with the additional burden of domain decomposition of the data, the leftover in-memory calculations are freely boostable with OpenMP, CUDA, or even a new layer of MPI, depending on the architecture of the computation units.

It is well known, that the achievable speedup on these methods heavily depends on the type of the problem. Is it possible to reach high performance in this special case? In this case the present local resources are investigated. Two many-CPU-core systems are available:

1. **MEMO** is an SMP NUMA system with 8 pieces of Intel Xeon X7560 processors @2.27GHz. These have 8 cores and 24 MByte of shared L3 cache each, connected with QPI, and additional 1 TByte of RAM.
2. **PLEXI** is a cluster system with 20 nodes. Each node is a NUMA system with 2 pieces of Intel Xeon X5650 processors @2.67GHz. These have 6 cores and 12 MByte of L3 cache each. Nodes are interconnected with quad-speed Infiniband and have 48 GByte of RAM.

As MEMO can handle up to 64 threads for one process, this machine will be used for OpenMP benchmarks. All other measurements are issued on PLEXI. The software environment includes GCC version 4.1.2 as C++ compiler with libgomp version 3.0 for OpenMP support, and OpenMPI 1.4.2 for MPI support.

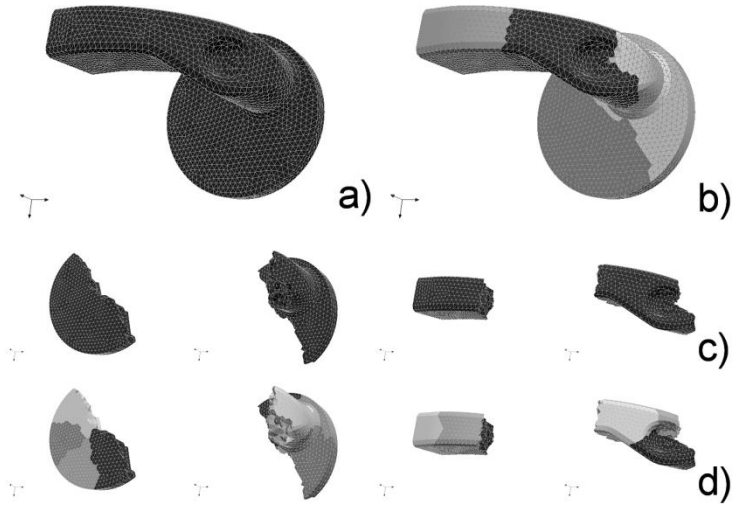

Figure 2. Visualization of various methods of parallelization. Single process, single thread, serial (a); single process multiple threads, OpenMP (b); multiple processes, one thread each, MPI (c); multiple processes, multiple threads each, Hybrid (d)

There are several issues in the simulation at hand, which makes parallelization a challenging task. Most of the vital issues, listed below, have been addressed, there are however several others, that need further attention.

- There are multiple types of meshes, each requiring unique treatment. This messes up domain decomposition. However, the support of handling multiple domains within one process, not just one, ensures manageability.
- There are interaction surfaces with more than two meshes connecting. However at a given time there are only two active cells on the opposing sides. Introducing an additional step, that synchronizes active cells only will prevent invalid data to be used.

Some of the non vital issues to be addressed presently are as follows:

- Inactivating, ergo skipping certain cells in the calculations induces tip over in the load balance. This is particularly present in a combustion chamber of high compressibility. Presently this is handled with special domain decomposition: the structured cylinder is split up along the axis of the piston motion. This ensures the nearly equal distribution of the computations. However communication costs are far from optimal.
- Special structured meshes, that require a wide stencil interpolation while snapping do not support decomposition within the stencil. This is considered a minor problem, as these meshes have few cells, the number of cells scales slower than linear, as the resolution gets finer.

The implementation is extensively tested to ensure the transparency of the MPI layer. Benchmarks are run for a quarter cycle, with the piston moving from the topmost to the bottommost position. Computation time is measured for each 200-loop-period, these are

treated as measured points. Speedup is calculated from this value by dividing the time cost of the serial code by that. Results for MPI, OpenMP and Hybrid measurements are shown on Figure 3, left.

Additional measurements are conducted with simpler, auto-generated meshes, in order to test scaling on the number of cells. Only one piston-like mesh is deforming in this model. Results on the strong scaling is shown on Figure 3, right.

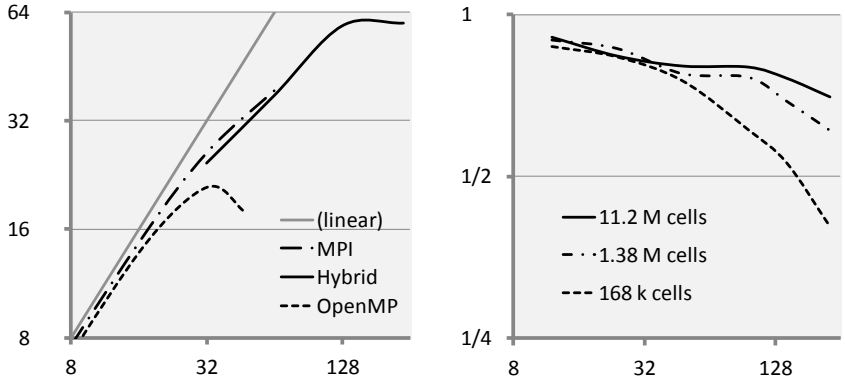

Figure 3. Achieved speedup values for the simulated combustion chamber with moving piston and valves is shown on the (left). The speedup is plot against the number of computing cores on log-log scale for the methods investigated. The strong scaling of the hybrid method is shown for generated systems with various cell counts (right). The speedup to number of cores ratio is plot against the number of cores on log-log scale.

It is observable, that the asymmetry resulting from the NUMA architecture does impair OpenMP implementation, that can be only improved by decomposition-like data aggregation. MPI scales very well up to a few processes per node, however fragmenting a not-so-large mesh to even smaller pieces will reveal a large surface to communicate. Also this lowers the available RAM per one process, which can inhibit the simulation from running. The MPI-OpenMP hybrid configuration with one process per processor (not node!) proves to achieve the best speedups. Running with above 128 cores, a speedup of around 60 is achievable, although there is still room for improvement.

We can also observe that the scaling is better with larger system size. This is naturally expected, as the communication cost proportionally lower with more cells to process. It can be stated, that this implementation is feasible for simulations of the presented kind with a cell count beyond one million up to ~200 processor cores.

3. Assembling multiple meshes

A rare case of challenge is presented in this part. For most FVM simulations mesh data is treated as one, partitioning and decomposition is done by extracting parts from the same mesh. In our case, however, there are special structured meshes, which are generated on the fly, from specific, marked surfaces of particular meshes. As most parts of the combustion chamber is deforming (the chamber itself, and meshes around all

valves as well), it must be ensured, that common surfaces of generated and unstructured meshes fit.

There are two problems, that make this fit harder. For once, mesh geometry data are stored in NASTRAN files [12], and the current PARMOD system supports only short field formats of 8 characters. This format may cut precision at even 10^{-4}, which is much cruder than the float precision of 10^{-7}. Additional scatter is introduced by human error. It is impossible to edit three dimensional data precisely as a machine.

Fitting the naïve way would require checking the distance of each pair of vertices. If we have *n* points on the surface, so *2n* points to be paired, time cost would be $O\ n^2$. Provided we have point pairs with coordinates that match exactly, ordering both set of coordinated lexicographically will place pairs in the same position.

The scatter in the coordinates renders this ordering unusable. A small change in the first coordinate can warp away the point to a far position. This is an optimization problem referred to as nearest neighbor search [13]. Usual solutions include space partitioning methods, like the k-d tree [14], and the local sensitive hashing, or "bucketing" [15]. The later one is implemented, and discussed here.

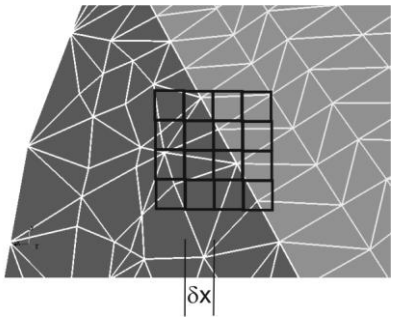

Figure 4. The joining surface of two meshes (structured on the left, unstructured on the right). The vertices of both meshes are "hashed" into buckets of linear size dx, and the shape of a cube. (8 different hashing are required for proper matching in three dimensions)

Using local sensitive hashing, each vertex is thrown in a cube form bucket of appropriate size, as shown on Figure 4. Two vertices in the same bucket can be considered coinciding. There is the possibility however, that two points sufficiently close to each other are not considered, as they sit in two separate, adjacent buckets. To avoid this, seven more bucketing is also conducted, and coincidence is approved, if any of the eight hashing gets two vertices in the same bucket.

The bucket has the actual size of the matching tolerance. This value has to be chosen carefully. Examining the spectrum of the inter-mesh vertex-to-vertex distance, there is a big gap below the minimum of the intra-mesh grid size. Human error will introduce a far smaller scatter, than this value. If the matching tolerance is chosen not far below that, the mesh joining will be successful. This will reduce matching time to $O(2^d n \log n)$. This method still slows down exponentially with the increasing of the dimensions. This is called the curse of dimensionality.

4. Mesh swap

Within the simulation at hand, there are several events, which require the change of discretization within the simulation space. First of them is the snapping in the simple deforming structured mesh, like the combustion chamber deformed by the moving piston. In this case, data in one or two cell layers gets interpolated. The next event is also snapping, now in the interface mesh, shown on Figure 5. This mesh handles interaction between two regions, where the surfaces are sliding on each other. During snapping, almost the entire mesh data is recalculated with interpolation.

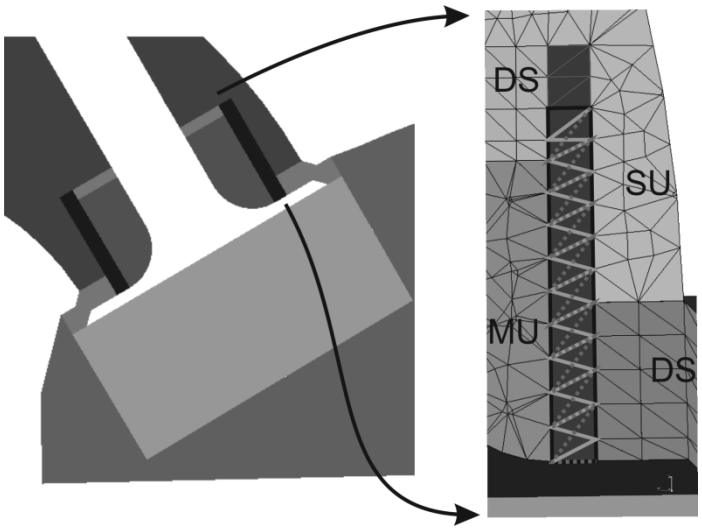

Figure 5. The mesh structure of the valve is shown on the (left). It consists of a moving unstructured (MU), three deforming structured (DS), and a special slipping interface mesh. It is engulfed by two static unstructured (SU) meshes on the top and bottom. The snapping of the interface mesh is shown on the (right). The discretization within the black frame changes from the continuous to the dotted, if the valve is closing (or vice versa, if it is opening).

The last case of mesh swap is in the case of the piston motion overlapping the path of the valves. This requires heavy data transformation with interpolation between multiple large meshes from different types. The parallel implementation of the later is also complicated, and yet to be done.

Interpolation must be implemented in a way, which is appropriate for the system simulated. It must preserve the corresponding physical properties:

- As all state variables are intensive, the weighed average must not change for any of the variables. A change in these variables will faultily create material, energy, or impulse.
- The local physical structure is to be conserved. Changing the mesh structure too often or with a too wide stencil will smear and degrade fine scale information.

In the sense of the above, data transformation is done in a way, where the new cell value is a weighed average of the data from the old cells that intersect with the new one. Weights are proportional to the relative common volume of the cells.

The first step is to calculate the sparse matrix with relative weights, showing the proportional volume coming from the intersection of the i^{th} and j^{th} cell to the space of the i^{th} cell in row *i*, column *j*.

The naïve algorithm for the calculation of the interpolation constants would require $O(nm)$ time. However, provided having convex cells, the old cells intersecting the new one will form a closed cluster. Thus using a breadth-first search on the neighbour structure, and junking non-intersecting old cells will yield a algorithm with a time cost proportional to the number of vertices: $O(\max(n, m))$.

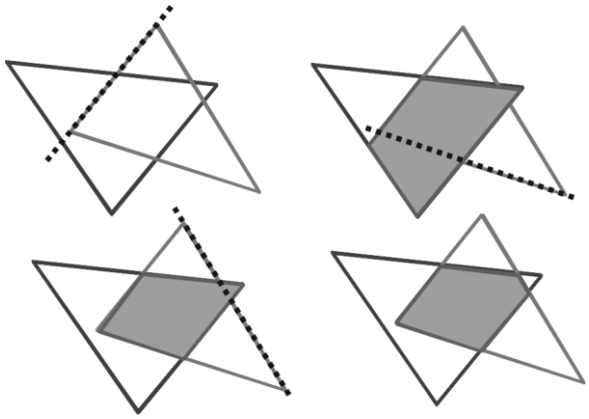

Figure 6. Step-by-step calculation of the intersection shape issuing one cutting plane at a time.

There is an algorithm to determine the volume of the intersection. The volume, *V*, of the intersection is calculated from the area, A_i, of the faces with the following formula:

$$V = \frac{1}{3} \quad _{face\ i}\, x_i n_i\, A_i, \tag{2}$$

where x_i is any point on the face, and n_i is the corresponding normal vector. Assuming the vertices within a face are on one plane, and they are indexed in order, the area, *A*, can be calculated as follows:

$$A = \frac{1}{2} \quad _i\ v_i - v_0\ \times (v_{i+1} - v_0)\ , \tag{3}$$

where v_ι is a vector of vertex coordinates.

The proper sorting approach will make interpolation time cost linear, making this approach ripe for runtime implementation.

5. Conclusion

It is apparent, that advanced computational techniques are required to handle sheer situations in this simulation, which can be especially strenuous for the average physicist.

It was also shown, that the proper use of these techniques can immensely aid the investigation of the physical phenomena. The speedup of 60 can reduce simulation time from five days to two hours. This allows for broader spectra of tests, or the investigation of a system that has almost four times the cell count.

Runtime usage of the mesh assembly and the interpolation procedures will greatly decrease manual work time on preparation. This also forecasts a more user-friendly like environment, hopefully usable by a wider range of users, like engineers.

There are still several challenging issues to be addressed. These include higher order, or implicit solution of the Euler equation within this setting. Also it would be most rewarding to implement acceleration for the graphics card with CUDA. The exhaustive study of moving mesh decomposition strategies and scaling would also prove invaluable.

These ideas will be realized next.

Acknowledgements

This work is done in conjunction with the project Simulation and Optimization on the Széchenyi István University, which is funded by the project TÁMOP-4.2.2-08/1-2008-0021 of the European Union. Major part of the software is based on previous works of András Horváth and Zoltán Horváth [10]. The simulation is using the PARMOD software system, primarily developed by Gábor Takács. Most meshes are created and generated by Tamás Morauszki, Péter Mándli and their team. To these people goes my sincere thanks. Special thanks to Gundolf Haase and Zoltán Horváth for the invaluable discussion on the topic.

References

[1] see e.g.: M. Lesoinne, C. Farhat: *Geometric conservation laws for flow problems with moving boundaries and deformable meshes, and their impact on aeroelastic computations*, Computer Methods in Applied Mechanics and Engineering **134,** 1996, pp 71-90.

[2] A. Jameson, W. Schmidt: *Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes*, AIAA **M,** 1981, pp 1-19.

[3] see e.g.: C. B. Laney: *Computational Gas Dynamics*, Cambridge University Press, 1998.

[4] P. H. Epstein et al.: *Computations of a Two-Stroke Engine Cylinder and port Scavenging Flow.* SAE 910672.

[5] J. Steger: *On the Use of Composite Grid Schemes in Computational Aerodynamics.* Computer Methods in Applied Mechanics and Engineering **64,** 1987, pp 301-320.

[6] A. A. Amsden et al.: *Comparisons of Computed and Measured Three-Dimensional Velocity Fields in a Motored Two-Stroke Engine.* SAE 920418.

[7] for the official version of the MPI standard see: http://www.mpi-forum.org/docs/docs.html

[8] for the official version of the OpenMP standard see: http://openmp.org/wp/openmp-specifications/
[9] for product description see: http://www.altairhyperworks.com/
[10] Z. Horváth, A. Horváth: *Numerical Simulation of Compressible Fluid Flow in 3D Domains with Translating Boundaries Proceedings in Applied Mathematics and Mechanics Vol. 4.* Issue 1., 2004, pp 422-423.
[11] L. Smith, M. Bull: *Development of mixed mode MPI / OpenMP applications*, Journal of Scientific Programming Vol. 9**.** Issue 2,3 2001.
[12] for extensive information on the file format, see: http://www.openchannelfoundation.org/projects/NASTRAN
[13] see e.g.: D. Knuth: *The Art of Computer Programming Vol. 3.* 1973.
[14] J. L. Bentley: *Multidimensional binary search trees used for associative searching, Communications of the ACM Vol. 18.* Issue 9. 1975.
[15] P. Indyk, R. Motwani: *Approximate nearest neighbors: towards removing the curse of dimensionality Proceedings of the thirtieth annual ACM symposium on Theory of computing* 604-613 1998.