FAST-EVP: Parallel High Performance Computing in Engine Applications

Gino Bella
Francesco Del Citto
Salvatore Filippone*
Faculty of Engineering,
University of Rome "Tor Vergata"
Viale del Politecnico, I-00133, Rome, Italy
*Corresponding author
E-mail: salvatore.filippone@uniroma2.it

Alfredo Buttari
Innovative Computing Lab, Department of Computer Science,
University of Tennessee at Knoxville,
1122 Volunteer Blvd. Knoxville, TN 37996, USA

Alessandro De Maio
N.U.M.I.D.I.A s.r.l.
Rome, Italy

Abstract: FAST-EVP (Fluid-Dynamics Analysis Software Tool for Engine Virtual Prototyping) is a simulation tool for internal combustion engines running on cluster platforms; it has evolved from the KIVA-3V code base, but has been extensively rewritten making use of modern numerical software for linear systems, parallel programming techniques and advanced physical models. The software is currently in use at the consulting firm NUMIDIA, and has been applied to a diverse range of test cases from industry, obtaining simulation results for complex geometries in short time frames.

Keywords: Computational Fluid Dynamics; Parallel Computing; Simulation

Biographical notes: Salvatore Filippone graduated in 1988 from the University of Rome "Tor Vergata"; he has always been active in the field of high performance computing. He has been with the IBM corporation from 1990 to 2000, where he worked in the development of numerical software for high performance architectures, and has joined the University of Rome "Tor Vergata" in 2001.

1 Introduction

The growing concern with the environmental issues and the request of reduced specific fuel consumption and increased specific power output are playing a substantial role on the development of automotive engines. Therefore, in the last decade, the automotive industry has undergone a period of great changes regarding the development methods for engine design, with an increased commitment to research. In particular, the use of advanced CFD techniques has been essential in achieving project targets in time while reducing product development costs. However, considerable work is still needed since CFD simulation of realistic industrial applications may take many hours or even weeks; this is a major problem in an industrial setting, given the short development times that are required to be competitive in today's marketplace.

The KIVA code [10] solves the complete system of general time-dependent Navier-Stokes equations and it is probably the most widely used code for internal combustion engine modeling. Its success mainly depends on its open source nature, which means having access to the source code. KIVA has been significantly modified and improved by researchers worldwide, especially in the development of sub-models to simulate the important physical processes that occurs in an internal combustion engine

(i.e. fuel-air mixture preparation and combustion); it uses a multi-block structured discretization mesh.

We have taken the KIVA code and its physical models as the base for the development of our simulation tool, called FAST-EVP (Fluid-Dynamics Analysis Software Tool for Engine Virtual Prototyping); this tool is currently in production use at NUMIDIA for consulting purposes.

The paper is organized as follows: in section 2 we review the basic mathematical model of the Navier-Stokes equations as discretized in our application, and its solution solution strategy. In section 3 we review the algorithmic issues that we confronted and describe the approach to the integration of new inner solvers with library routines from [6, 7], whereas in section 4 we outline the parallelization strategy. Finally we present some experimental results in section 5, followed by our conclusions. A preliminary account of the development work had been given in the conference proceedings [3]; here we complement that presentation with details about the treatment of valves and spray dynamics, and we present performance results for a reference engine test case obtained on an up-to-date computing platform.

2 Mathematical model

The mathematical model of KIVA-3 is the complete system of general *unsteady Navier-Stokes equations*, coupled with chemical kinetic and spray droplet dynamic models. In the following the equations for fluid motion are reported.

- Species continuity:

$$\frac{\partial \rho_m}{\partial t} + \nabla \cdot (\rho_m \mathbf{u}) = \nabla \cdot [\rho D \nabla (\frac{\rho_m}{\rho})] + \dot{\rho}^c_m + \dot{\rho}^s_m \delta_{ml} \quad (1)$$

  where ρ_m is the mass density of species m, ρ is the total mass density, $\mathbf{u}$ is the fluid velocity, $\dot{\rho}^c_m$ is the source term due to chemistry, $\dot{\rho}^s_m$ is the source term due to spray and δ is the Dirac delta function.

- Total mass conservation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = \dot{\rho}^s \quad (2)$$

- Momentum conservation:

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\,\mathbf{u}) = -\frac{1}{\alpha^2} \nabla p - A_0 \nabla (\frac{2}{3} \rho k) + \nabla \cdot \overline{\sigma} + \mathbf{F}^s + \rho \mathbf{g}$$

  where $\overline{\sigma}$ is the viscous stress tensor, $\mathbf{F}^s$ is the rate of momentum gain per unit volume due to spray and $\mathbf{g}$ is the constant specific body force. The quantity A_0 is zero in laminar calculations and unity when turbulence is considered.

- Internal energy conservation:

$$\frac{\partial (\rho I)}{\partial t} + \nabla \cdot (\rho I \mathbf{u}) = -p \nabla \cdot \mathbf{u} + (1 - A_0) \overline{\sigma} : \nabla \mathbf{u} - \nabla \cdot \mathbf{J} + A_0 \rho \epsilon + \dot{Q}^c + \dot{Q}^s$$

  where I is the specific internal energy, the symbol : indicates the matrix product, $\mathbf{J}$ is the heat flux vector, $\dot{Q}^c$ and $\dot{Q}^s$ are the source terms due to chemical heat release and spray interactions.

Furthermore the standard $k - \epsilon$ equations for the turbulence are considered, including terms due to interaction with spray [12].

2.1 Numerical method

The numerical method employed in KIVA-3 is based on a variable step *implicit Euler* temporal finite difference scheme, where the time steps are chosen using accuracy criteria. Each time step defines a cycle divided in three phases, corresponding to a physical splitting approach. In the first phase, spray dynamic and chemical kinetic equations are solved, providing most of the source terms; the other two phases are devoted to the solution of fluid motion equations [1]. The spatial discretization of the equations is based on a *finite volume method*, called the *Arbitrary Lagrangian-Eulerian method* [13], using a mesh in which positions of the vertices of the cells may be arbitrarily specified functions of time. This approach allows a mixed Lagrangian-Eulerian flow description. In the Lagrangian phase, the vertices of the cells move with the fluid velocity and there is no convection across cell boundaries; the diffusion terms and the terms associated with pressure wave propagation are implicitly solved by a modified version of the SIMPLE (Semi Implicit Method for Pressure-Linked Equations) algorithm [11]. This algorithm, well known in the CFD community, is an iterative procedure to compute velocity, temperature and pressure fields. Upon convergence on pressure values, implicit solution of the diffusion terms in the turbulence equations is approached.

3 Algorithmic issues

One of the main objectives in our work on the KIVA code [6] was to show that general purpose solvers, based on up-to-date numerical methods and developed by experts, can be used in specific application codes, improving the quality of numerical results and the flexibility of the codes as well as their efficiency.

The original KIVA code employs the Conjugate Residual method, one member of the Krylov subspace projection family of methods [14]. These methods have become widely used during the 1980s, and many new variants have been developed, especially for dealing with nonsymmetric systems. Our work thus started with the idea of introducing new linear system solvers and more sophisticated preconditioners in the KIVA code; to do this we had to tackle a number of issues related to the code design and implementation.

The KIVA code is based on a finite-volume discretization of the the simulation domain using hexahedral cells. The scalar quantities (such as temperature, pressure and

turbulence parameters), are evaluated at the centers of the cells, whereas the velocity is evaluated at the vertices of the cells. The cells are represented through the coordinates of their vertices; the vertex connectivity is stored explicitly in a set of three connectivity arrays, from which it is possible by repeated lookup to identify all neighbours. The original implementation of the CR solver for linear systems employs a *matrix-free* approach, i.e. the coefficient matrix is not formed explicitly, but its action is computed in an equivalent way whenever needed.

The major advantage of a matrix-free implementation is in terms of memory occupancy. However it constraints the kind of preconditioners that can be applied to the linear system; in particular, it is difficult to apply preconditioners based on incomplete factorizations. Given the above analysis, we integrated the solvers in [7] employing the following design criteria:

1. The solver routines should be separated into different phases: matrix generation, matrix assembly, preconditioner computation and actual system solution;
2. The matrix generation phase requires an user supplied routine that generates (pieces of) the rows of the matrix in the global numbering scheme, according to a simple storage scheme, i.e. coordinate format;
3. The data structures used in the solvers should be parametric, and well separated from those used in the rest of the application.

3.1 Integration of the numerical library

The basic groundwork for the parallelization and integration of the numerical library has been laid out at the time of [6], which we briefly review below.

The equations for thermodynamic quantities, pressure, temperature and turbulence, are written at cell centers, and they give rise to non-symmetric matrices a symmetric sparsity pattern, having no more than 19 nonzero entries per row. For the velocity equation, the discretization scheme, using a staggered grid, leads to non-symmetric coefficient matrices with no more than 27 entries per (block) row, where each entry is a 3×3 block.

The Conjugate Residual method used in the original code has no guarantee of covergence on non-symmetric matrices such as the ones we encounter in KIVA [14]; therefore we searched alternative solution and preconditioning methods. After suitable experimentation we settled on the Bi-CGSTAB method [14] for all of the linear systems in the SIMPLE loop, employing a block ILU preconditioner on the pressure correction equation; this resulted in a vast improvement in the code behaviour, which we analyzed in [6].

Further research work on other preconditioning schemes is currently ongoing, and we plan to include its results in future versions of the code [4, 5]; the preliminary tests have shown encouraging results in the convergence properties, and will enlarge the available tools in the search for the optimal solution technique.

4 Parallelization issues and new developments

Since the time of [6] the code has undergone a major restructuring: FAST-EVP is based on the KIVA-3V version, and thus it is able to model valves. This new modeling feature has no direct impact on the SIMPLE solvers interface, but affects the way we handle mesh movement changes. While working on the new KIVA-3V code base, we also reviewed all of the space allocation requirements, cleaning up a lot of duplications; in short we have now an application fully parallelized, even in its darkest parts.

All computations in the code are parallelized with a domain decomposition strategy: the computational mesh is partitioned among the processors participating in the computation. To perform this domain partition we employ the Metis [8] graph partitioning software; more details will be given in section 4.2.

The support library routines allow us to manage the necessary data exchanges throughout all phases of the code using the same data structures employed for the linear system solvers; thus, we have a unifying framework for all computational phases.

A very important part of the computation is the rezoning phase; here we adjust the grid points following the application of the fluid motion field; the algorithm is an explicit calculation that is based on the same "gather" and "scatter" stencils found in the matrix-vector products for the linear systems. It is thus possible to implement in parallel the explicit algorithm by making use of the boundary data exchange operations defined in the support library [7]. The data structures needed to perform these boundary data exchanges are a by-product of the parallelization of the sparse linear solvers, and given the isomorphism between the matrix sparsity pattern and the discretization mesh they can be employed for all mesh-related data exchanges, as exemplified by the spray droplet movements discussed below.

The rezoning algorithm has been made completely explicit for parallelization purposes, that is while cycling on grid nodes only the coordinates of the current node are changed, taking into account the positions of neighbours and the distances from moving or solid surfaces in all the six directions. The algorithm will perform multiple sweeps over the mesh, with convergence being declared when every node of the mesh moves less than a fixed threshold.

Due to the complex geometries of modern engines, it is almost impossible to find a set of weights for the weighted mean that could guarantee a good quality of the grid for all nodes, regardless of the position of moving surfaces. Hence it is necessary to define various portions of logical grid, even for different crank angles, and move the nodes included in them with different values of the weights.

All the data needed by the rezoning algorithm can be specified in the input file, as opposed to traditional development in which the various parameters involved were hardcoded into the rezoning subroutines; to achieve this user–friendly structure for input and obtaining performaces similar to a compiled routine, we employed the struc-

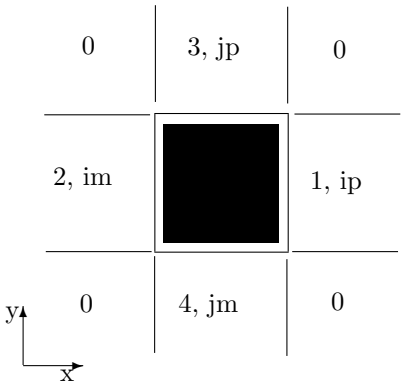

Figure 1: Zones around valve stem for rezoning algorithm

tured data types and dynamic memory management features of Fortran 95.

The input blocks will also contain the parameters necessary to model movement of nodes around the valve stems; furthermore, for each valve we can specify values for the weigths in each of the five zones around the valve stem, depicted in figure 1, thus providing the necessary flexibility to model complex configurations.

4.1 Spray dynamics

For the spray dynamics model we had to implement specific operators that follow the spray droplets in their movement, transferring the necessary information about the droplets whenever their simulated movement carries them across the domain partition boundaries. These operators are based on the same communication data structures used in the linear system solution and mesh handling.

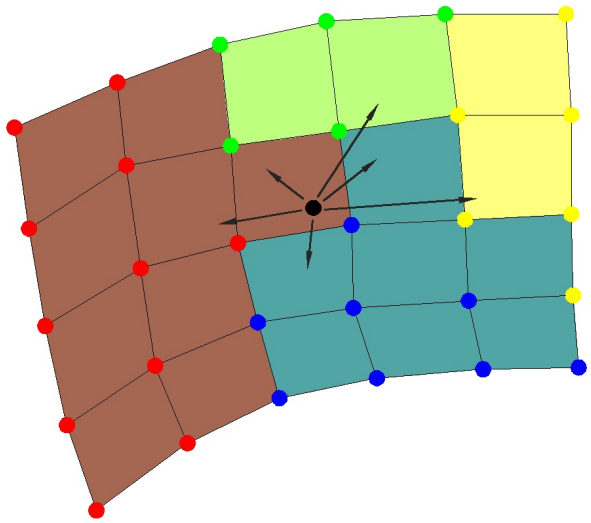

Figure 2: Droplets movement

The droplets are followed through their movement across domain boundaries in the following way (see figure 2):

1. Droplets are assigned to the process that owns the cell in which they are located at each time step;
2. At each time step a droplet is moved with its own velocity and direction;
3. If the droplet movement is such that the droplet ends up in a cell that is owned by the same process as the cell from which it is started, no communication is necessary;
4. If a droplet ends up in a cell that is adjacent to the boundary of the domain assigned to the processor, then it is already known which process owns that cell, because of the communication routines that exchange data for the field solvers; thus the droplet is added to the list of data to be exchanged
5. If, after all droplets have been processed and neighbouring communications have been taken care of, there still are "orphan" droplets, their coordinates are broadcast to all processes, and each process will start a location algorithm to find out if they ended up in a local cell; this is the case shown in figure 2 for the droplet that crosses two domain boundaries.

Special attention had to be paid upon injection, to find the topological position of a droplet after a time-step; this is in principle a very time consuming task, because we essentially look up the geometric position for a particle and then try to figure out the index of the cell containing it, and thus we have to be careful in order to reduce the computing time.

Another issue that needed a careful treatment was the (possible) change of owner process for a fuel film region across a snap event, something that can happen, especially when the film lies on a moving surface.

Depending on the type of partition of the computational domain and on the injection conditions our strategy may in some cases create a computational imbalance, but this is not a common problem, and is usually limited to a very small section of the simulation.

4.2 Mesh movement

The simulation process modifies the finite volume mesh to follow the (imposed) piston and valve movement during the engine working cycle; two different mesh configurations for one of our test cases may be seen in figure 3. The computational mesh is first deformed by reassigning the positions of the finite volume surfaces in the direction of the piston movement, until a critical value for the cell aspect ratio is reached; at this point a layer is cut out from (or added into) the mesh to keep the aspect ratio within reasonable limits. When this "snapper" event takes place it is necessary to repartition the mesh and to recompute the patterns of the linear system matrices.

In the original version of Kiva, the piston was considered topologically flat, while a bowl below the piston surface was marked and treated separately. At present state of

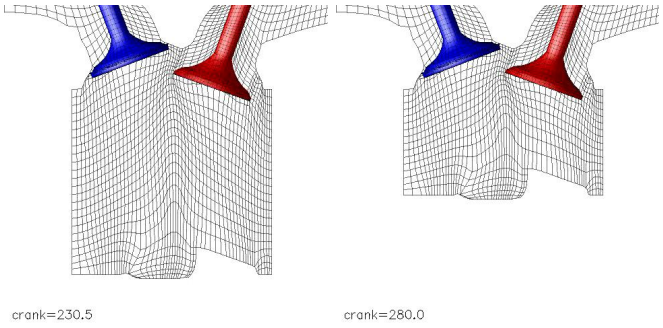

Figure 3: Mesh movement

development, the definition of bowl has been completely eliminated, while the piston surface could have any required topological profile. Hence, the valves can now move through any fluid cell in the squish zone, even below the original flat piston; the only remaining limitation is requirement for the presence of two or more fluid cells planes between two any surfaces.

Similarly, the movement of valves is monitored and additional "snapper" events are generated accordingly to their opening or closing. In the original code, during valve displacement, the nodes on the stem were not moved at all, while only the head of the valve was involved in movement and snap events. This could lead to deformations of the shape of the valve, as shown in figure 4. Thus we implemented a new strategy for moving valves: we treat a valve in its entirety, including the stem, as a rigid object, and we insert or remove node planes from the top of the stem, instead of the top of the head. The result of this approach is shown in figure 5, where we can see that the new approach requires an additional rezoning algorithm around the stem.

Across a "snapper" event it is necessary to recompute the data distribution for the parallel data structures; this is done with a hybrid strategy to minimize the number of repartitionings. At the beginning of the application we invoke the Metis graph partitioning tool [8] on the whole discretization mesh. Each subsequent snapper event may either add or delete a cell layer above the piston head. Immediately after the addition of a cell layer we extend the previous partition by assigning each new cell to the same process as the cell immediately above it; if instead we had a layer deletion, we simply keep the other cells' assignments. We then compute an estimate of the load imbalance obtained by this simple strategy; thus we only invoke a repartition of the mesh if the load imbalance is beyond a threshold chosen by the user. When spray is included in the simulation, the computational load is altered by the spray droplets; we have seen in the previous section that these droplets move at each time step, therefore we also need to evaluate the load imbalance taking into account the droplet position after each time step during the injection phase, and possibly do a repartition.

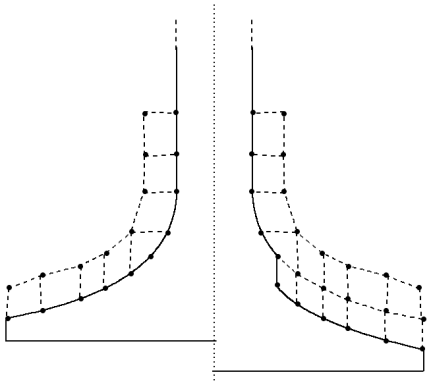

Figure 4: Deformed valve after an original snap event.

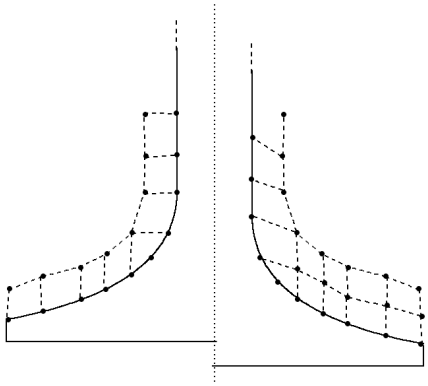

Figure 5: New valve snap strategy.

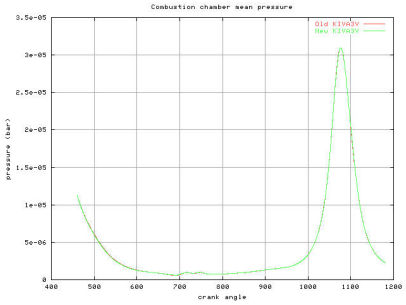

Figure 7: Competition engine average pressure

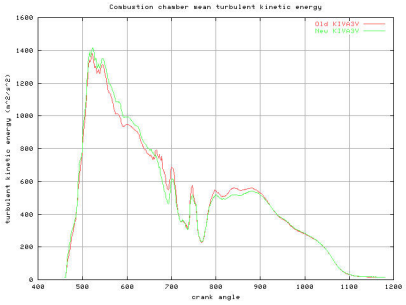

Figure 8: Competition engine average turbulent energy

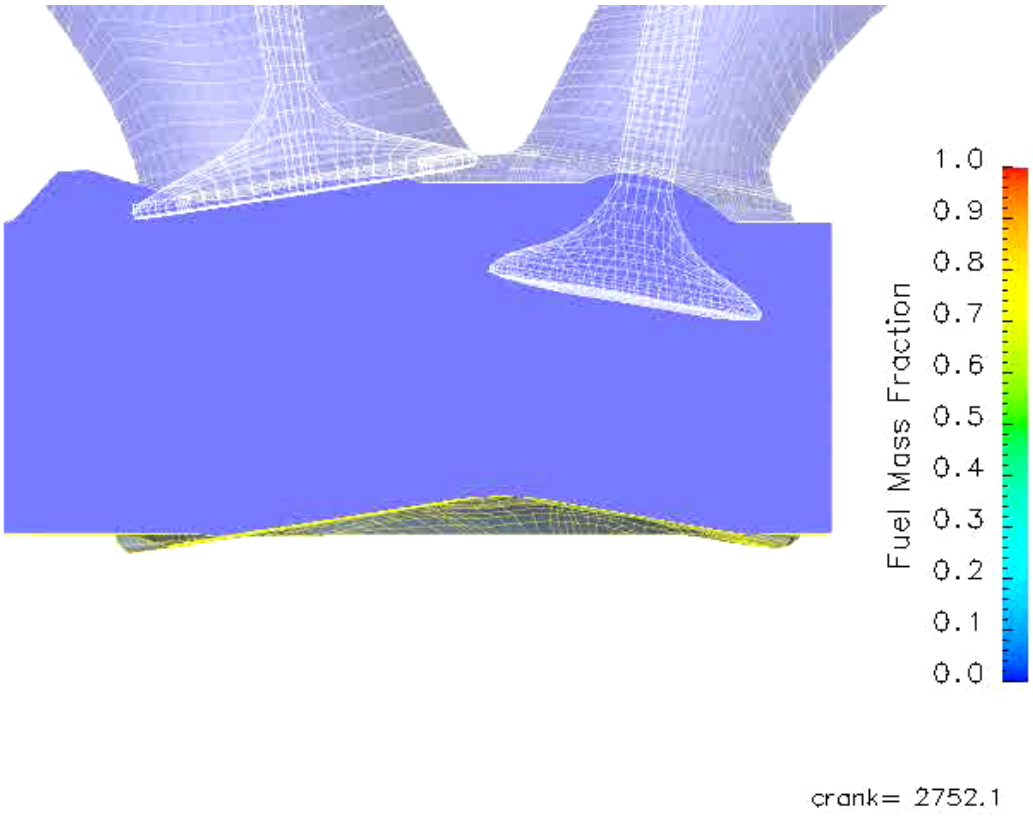

Figure 6: Competition engine simulation

5 Experimental results

The experimental results presented in this paper are based on a high performance competition engine that was used to calibrate our software. The choice of this engine was due to the availability of measurements to compare against, so as to make sure not to introduce any modifications in the physical results; moreover it is a very demanding and somewhat extreme test case, because of the high rotation regime, high pressure injection conditions, and relatively small mesh size.

A cross section of the mesh, immediately prior to the injection phase, is shown in figure 6; the overall mesh is composed of approximately 200K control volumes. The computation has been carried out on a Linux cluster available at the computing center of CASPUR (Inter-University Consortium for the Application of Super-Computing for Universities and Research) in Rome, comprising dual-processor nodes based on the AMD Opteron 250 with a 4 GB RAM and a 1024 KB cache memory. The nodes are connected via InfiniBand Silverstorm InfiniHost III Ex HCA; this network interface has a user level latency of approx. 5 μsec and a measured sustained bandwidth of 960 MB/sec.

The partitioning of the mesh on 16 processes is shown in figure 9; the snapshot is taken close to the top dead center. The test runs with combustion activated were performed for a fixed interval of crank angle around the ignition point,

# P	Time (m)	Speedup	Cycles	T/cy. (m)	Sp./cy
1	114.04	1.00	90	1.27	1.00
2	66.64	1.71	92	0.72	1.75
4	49.87	2.29	96	0.52	2.44
6	44.83	2.54	121	0.37	3.42
8	22.28	5.12	90	0.25	5.12
12	18.36	6.21	103	0.18	7.11
16	12.17	9.37	90	0.14	9.37
20	9.94	11.47	90	0.11	11.47
24	8.73	13.06	90	0.10	13.06
30	6.02	18.94	90	0.07	18.94
36	6.18	18.45	90	0.07	18.45
42	5.23	21.80	90	0.06	21.80

Table 1: Test case timing data

corresponding to approximately 90 time steps. Snapshots of the temperature and burned mass fraction distributions are shown in figure 10 and 11.

First of all we note that the global physical quantities typically required to evaluate the overall engine performance, such as average pressure and turbulent kinetic energy, are in line with those obtained by the original serial code, as shown in figure 7 and 8; multiple comparisons have been performed, with different number of processors, to confirm that indeed we are computing physically consistent results. This is true even if it is impossible to obtain

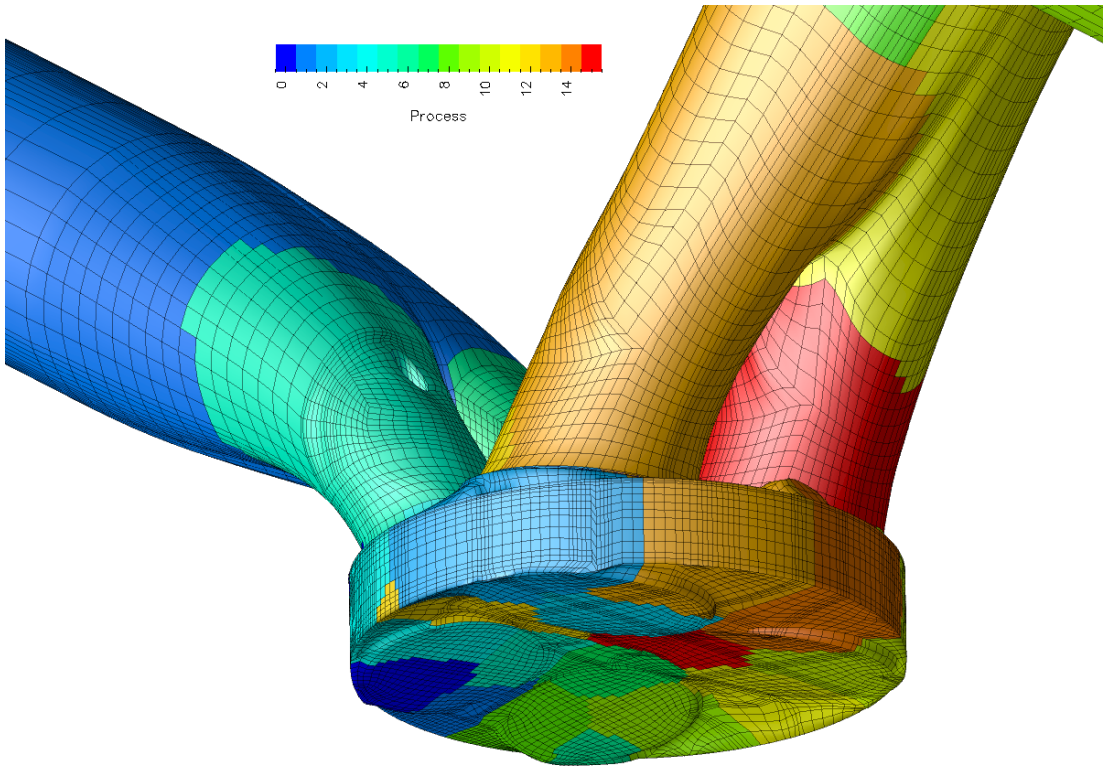

Figure 9: Mesh partitioning on 16 processes

identical results for runs made on different machine configurations, due to variation in ordering of floating point operations induced by the parallelization process.

In table 1 we report the timings and speedup data, overall and per time-step, for the simulation on different parallel machine configurations. The scalability of the application is very satisfactory, even if the computational mesh is not very large; as the number of processes grows we should expect the utilization to decrease when using a fixed mesh, because the balance between computation and communication exchanges is altered, yet the behaviour on this machine is very good even at 42 processes, with a parallel efficiency over 50 %. It should be noted explicitly that in normal usage larger machine configurations would be employed when performing more detailed and demanding simulations on larger discretization meshes. The overall speedup and efficiency at 6 and 12 processes deteriorates because of the increase in the number of time-steps needed to cover the same crank angle interval; this is due to the effects of the domain partitioning on the convergence of the linear solver.

6 Conclusion

We have discussed FAST-EVP, a new engine design application based on an extensive revision and rewrite of the KIVA-3V application code, and parallelized by use of the PSBLAS library. The application has been tested on industrial test cases, and has proven to be robust and scalable, enabling access to results that were previously impossible.

Future development work includes further refinement of the explicit rezoning and spray dynamics phases, and experimentation with new preconditioners and linear system solvers.

Acknowledgments

The authors wish to thank the anonymous reviewers for their helpful comments on the paper draft.

REFERENCES

[1] A.A. Amsden, P.J. O'Rourke, T.D. Butler, *KIVA-II: A Computer Program for Chemically Reactive Flows with Sprays*, Los Alamos National Lab., Tech. Rep. LA-11560-MS, 1989.

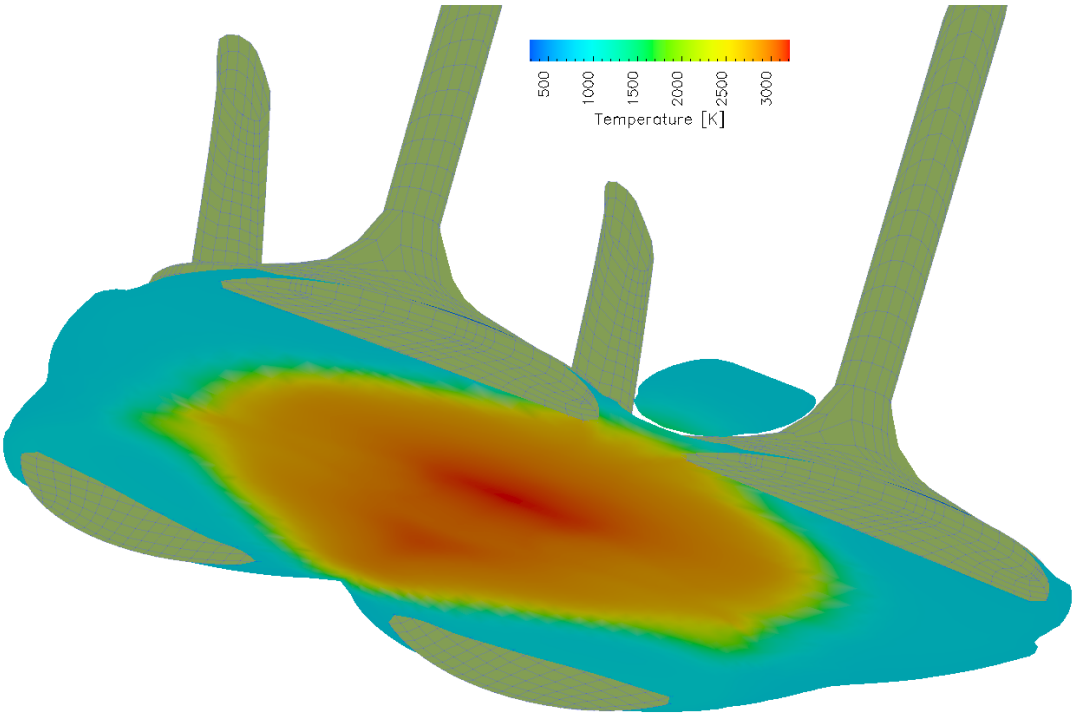

Figure 10: Fluid temperature inside combustion chamber, top view

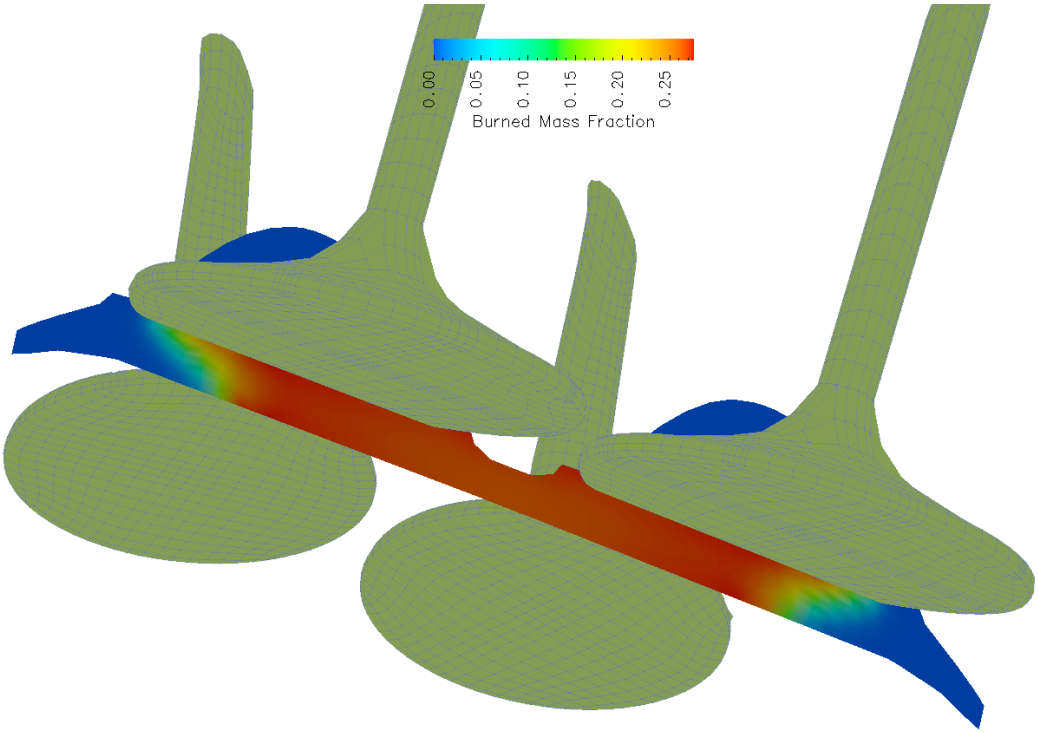

Figure 11: Burned mass fraction inside combustion chamber, side view

[2] A.A. Amsden, *KIVA 3: A KIVA Program with Block Structured Mesh for Complex Geometries*, Los Alamos National Lab., Tech. Rep. LA-12503-MS, 1993.

[3] G. Bella, A. Buttari, A. De Maio, F. Del Citto, S. Filippone and F. Gasperini: *FAST-EVP: An Engine Simulation Tool*, in High Performance Computing and Communications, 1st International Conference, HPCC 2005, PARA 2004, Springer Lecture Notes in Computer Science 3726, pp. 979–990, 2005.

[4] A. Buttari, P. D'Ambra, D. di Serafino and S. Filippone: *Extending PSBLAS to Build Parallel Schwarz Preconditioners*, in Applied Parallel Computing. State of the Art in Scientific Computing: 7th International Conference, PARA 2004, Springer Lecture Notes in Computer Science 3732, pp. 593–602, 2006.

[5] P. D'Ambra, D. Di Serafino, and S. Filippone: *On the Development of PSBLAS-based Parallel Two-level Schwarz Preconditioners*, Applied Numerical Mathematics, to appear, 2006.

[6] S. Filippone, P. D'Ambra, M. Colajanni: *Using a Parallel Library of Sparse Linear Algebra in a Fluid Dynamics Applications Code on Linux Clusters.* Parallel Computing - Advances & Current Issues, G. Joubert, A. Murli, F. Peters, M. Vanneschi eds., Imperial College Press Pub. (2002), pp. 441–448.

[7] S. Filippone and M. Colajanni. PSBLAS: A library for parallel linear algebra computation on sparse matrices. *ACM Trans. Math. Softw.*, 26(4):527–550, December 2000.

[8] Karypis, G. and Kumar, V. *METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering System.* Minneapolis, MN 55455: University of Minnesota, Department of Computer Science. Internet Address: `http://www.cs.umn.edu/~karypis`.

[9] C. T. Kelley: Iterative Methods for Linear and Nonlinear Equations, SIAM, 1995.

[10] P.J. O'Rourke, A.A. Amsden, *Implementation of a Conjugate Residual Iteration in the KIVA Computer Program*, Los Alamos National Lab., Tech. Rep. LA-10849-MS, 1986.

[11] S.V. Patankar, *Numerical Heat Transfer and Fluid Flow*, Hemisphere Publ. Corp.

[12] S. B. Pope, *Turbulent Flows*, Cambridge Univ. Press, 2000.

[13] W.E. Pracht, *Calculating Three–Dimensional Fluid Flows at All Speeds with an Eulerian–Lagrangian Computing Mesh*, J. of Comp. Physics, Vol.17, 1975.

[14] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd ed., SIAM, Philadelphia, 2003.