FAST-EVP: An Engine Simulation Tool

Gino Bella, Alfredo Buttari, Alessandro de Maio, Francesco del Citto, Salvatore Filippone, Fabiano Gasperini

FAST-EVP: an Engine Simulation Tool

Gino Bella[1], Alfredo Buttari[1],Alessandro De Maio[2], Francesco Del Citto[1],Salvatore Filippone[1], Fabiano Gasperini[1]

[1] Faculty of Engineering,
Università di Roma "Tor Vergata"
Viale del Politecnico, I-00133, Rome, Italy
[2] N.U.M.I.D.I.A s.r.l.
Rome, Italy

Abstract. FAST-EVP 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 linear solvers, 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.

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 industries have undergone a period of great changes regarding the engines design and development methods with an increased commitment to research by the industry. In particular, the use of CFD have revealed to be of great support for both the design and the experimental work in order to quickly achieve the projects targets while reducing the product development costs. However, considerable work is still needed since CFD simulation of realistic industrial applications may take many hours or even weeks that not always agrees with the very short development times that are required to a new project in order to be competitive in the market.

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 engines 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).

In the following we will review the basic mathematical model of the Navier-Stokes equations as discretized in our application; we will describe the approach

to the linear system solution based on the library routines from [7, 8]. In particular we outline the new developments in the spray dynamics and explicit flux phases that resulted in the code being used today; finally, we show some experimental results.

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:

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

  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:

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

- Momentum conservation:

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

  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:

$$\frac{\partial (\rho I)}{\partial t} + \nabla \cdot (\rho I \mathbf{u}) = -p \nabla \cdot \mathbf{u} + (1 - A_0) \overline{\sigma} : \nabla \mathbf{u} - \nabla \cdot \mathbf{J} + A_0 \rho \epsilon + \dot{Q}^c + \dot{Q}^s \tag{4}$$

  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.

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* [16], 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 [15].Upon convergence on pressure values, implicit solution of the diffusion terms in the turbulence equations is approached. Finally, explicits methods, using integral sub-multiple time-steps of the main computational time step, are applied to solve the convective flow in the Eulerian phase.

3 Algorithmic issues

One of the main objectives in our work on the KIVA code [7] 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 [3, 12, 13, 17]. Krylov subspace methods originate from the Conjugate Gradient algorithm published in 1952, but they became widely used only in the early 80s. Since then this field has witnessed many advances, and many new methods have been developed especially for non-symmetric linear systems. The rate of convergence of any given iterative method depends critically on the eigenvalue spectrum of the linear system coefficient matrix. To improve the rate of convergence it is often necessary to *precondition* it, i.e. to transform the system into an equivalent one having better spectral properties. Thus our work 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.

3.1 Code design issues

KIVA-3 is a finite-volume code in which the simulation domain is partitioned into hexahedral cells, and the differential equations are integrated over the cell to obtain the discretized equation, by assuming that the relevant field quantities are constant over the volume. 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

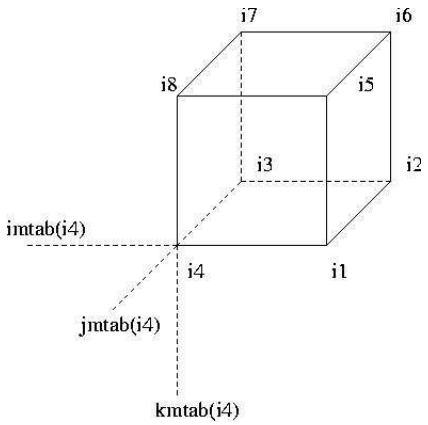

Fig. 1. Vertex numbering for a generic control volume

to identify all neighbours, as shown in figure 1. The original implementation of the CR solver for linear 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; this is done by applying the same physical considerations that would be needed in computing the coefficients: there is a main loop over all cells, and for each cell the code computes the contribution from the given cell into the components corresponding to all adjacent cells (including itself), from $i1$ through $i8$. This is a "scatter" approach, quite different from the usual "gather" way of computing a matrix-vector product.

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. Moreover, from an implementation point of view, data structures strictly based on the modeling aspects of the code do not lend themselves readily to transformations aimed at achieving good performance levels on different architectures.

The above preliminary analysis has influenced the design of the interface to the sparse linear solvers and support routines in [8] with the following characteristics:

1. The solver routines are well 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 are parametric, and well separated from those used in the rest of the application.

4 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 [7], which we briefly review below.

The code to build the matrices coefficients has been developed starting from the original solver code: the solvers in the original KIVA code are built around routines that compute the residual $r = b - Ax$, and we started from these to build the right hand side b and the matrix A. Since the solution of equations for thermodynamic quantities (such as temperature, pressure and turbulence) requires cell center and cell face values, the non-symmetric linear systems arising from temperature, pressure and turbulence equations have coefficient matrices with the same symmetric sparsity pattern, having no more than 19 nonzero entries per row. In the case of the linear systems arising from the velocity equation, following the vectorial solution approach used in the original code, the unknowns are ordered first with respect to the three Cartesian components and then with respect to the grid points. The discretization scheme leads to non-symmetric coefficient matrices with no more than 27 entries per (block) row, where each entry is a 3×3 block.

4.1 Algorithmic Improvements

The original KIVA-3 code solution method, the Conjugate Residual method, is derived under the hypothesis of a symmetric coefficient matrix; thus, there is no guarantee that the method should converge on non-symmetric matrices such as the ones we encounter in KIVA. Therefore we went to search for alternative solution and preconditioning methods. Since the convergence properties of any given iterative method depend on the eigenvalue spectrum of the coefficient matrices arising in the problem domain, there no reason to expect that a single method should perform optimally under all circumstances [11, 14, 17]. Thus we were led to an experimental approach, in searching for the best compromise between preconditioning and solution methods. We settled on the Bi-CGSTAB method for all of the linear systems in the SIMPLE loop; the critical solver is that for the pressure correction equation, where we employed a block ILU preconditioner, i.e. an incomplete factorization based on the local part of A. The BiCGSTAB method always converged, usually in less than 10 iterations, and practically never in more than 30 iterations, whereas the original solver quite often would not converge at all.

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].

5 Parallelization issues and new developments

Since the time of [7] the code has undergone a major restructuring: FAST-EVP code is now 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 it is important in handling 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. This partitioning is induced by the underlying assumptions in the linear system solvers; however it is equally applicable to the rezoning phase. The support library routines allow us to manage the necessary data exchanges throughout the code based on the same data structures employed for the linear system solvers; thus, we have a unifying framework for all computational phases.

The rezoning phase is devoted to adjusting 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 phase in Fig. 1. It is thus possible to implement in parallel the explicit algorithm by making use of the data movement operations defined in the support library [8].

The chemical reaction dynamics is embarassingly parallel, because it treats the chemical compounds of each cell independently.

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 brings them across the domain partition boundaries.

5.1 Mesh movement

The simulation process modifies the finite volume mesh to follow the (imposed) piston and valve movement during the engine working cycle (see also Fig. 2. 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 (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. The algorithm for matrix assembly takes into account the above considerations by preserving the matrix structure between two consecutive "snapper" points, and recomputing only the values of the non-zero entries at each invocation of the linear system solver; this is essential to the overall performance, since the computation of the structure is expensive.

Similarly, the movement of valves is monitored and additional "snapper" events are generated accordingly to their opening or closing; the treatment is completely analogous to that for the piston movement.

6 Experimental results

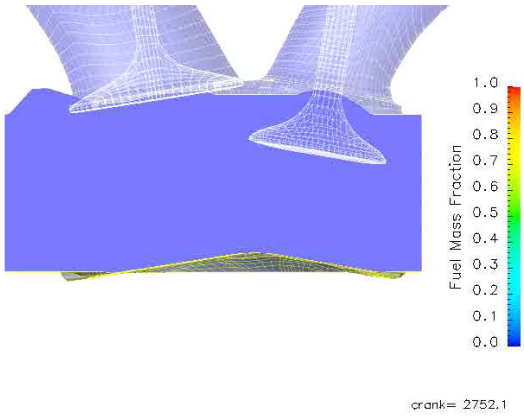

Fig. 2. Competition engine simulation

Processes	Time steps	Total time (min)
1	2513	542
2	2515	314
4	2518	236
5	2515	186
6	2518	175
7	2518	149

Table 1. Competition engine timings

The first major test case that we discuss is 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 section of the mesh, immediately prior to the injection phase, is shown in Fig. 2; the overall mesh is composed of approximately 200K control volumes. The simulated comprises 720 degrees of crank angle at 16000 rpm, and the overall timings are shown in Table 1. The computation has been carried out at NUMIDIA on a cluster based on Intel Xeon processors running at 3.0 GHz,

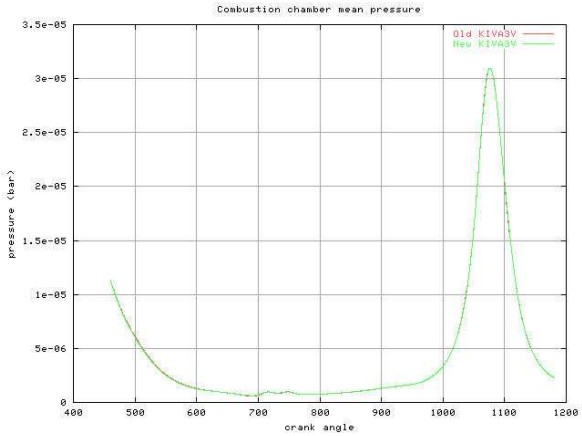

Fig. 3. Competition engine average pressure

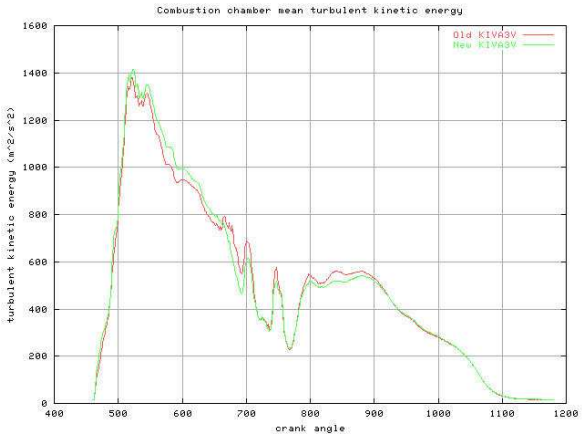

Fig. 4. Competition engine average turbulent energy

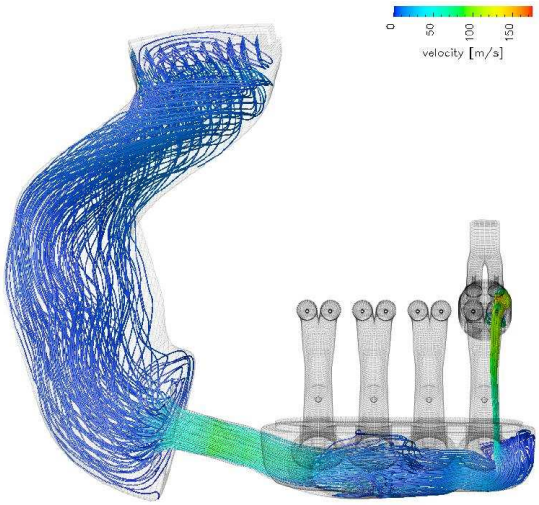

Fig. 5. Commercial engine air flow results

equipped with Myrinet M3F-PCIXD-2 network connections. The physical results were confirmed to be in line with those obtained by the original code, as shown in fig. 3 and 4.

Another interesting test case is shown in Fig. 5; here we have a complete test of a commercial engine cylinder coupled with an air box, running at 8000 rpm, of which we detail the resulting airflow. The discretization mesh is composed of 483554 control volumes; the simulation comprises the crank angle range from 188 to 720 degrees, with 4950 time steps, and it takes 703 minutes of computation on 9 nodes of the Xeon cluster. In this particular case the grid had never been tested on a serial machine, or on smaller cluster configurations, because of the excessive computational requirements; thus the parallelization strategy has enabled us to obtain results that would have been otherwise unreachable.

7 Conclusion

We have discussed 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 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.
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. R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donat, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the solution of linear systems.* SIAM, 1993.
4. A. Buttari, P. D'Ambra, D. di Serafino and S. Filippone: *Extending PSBLAS to Build Parallel Schwarz Preconditioners*, in Proceedings of PARA'04, to appear.
5. P. D'Ambra, D. Di Serafino, and S. Filippone: *On the Development of PSBLAS-based Parallel Two-level Schwarz Preconditioners*, Preprint n. 1/2005, Dipartimento di Matematica, Seconda Università di Napoli, 2005.
6. P. D'Ambra, S. Filippone, P. Nobile, *The Use of a Parallel Numerical Library in Industrial Simulations: The Case Study of the Design of a Two-Stroke Engine*, Proc. of Annual Meeting of ISCS'97, Istituto Motori-CNR ed., 1998.
7. 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.
8. 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.
9. J. Dongarra and R. Whaley, *A user's guide to the BLACS v1.0.* LAPACK working note #94 (June), University of Tennessee, 1995. `http://www.netlib.org/lapack/lawns`.
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. N. M. Nachtigal, S. C. Reddy, L. N. Threfethen: How fast are nonsymmetric matrix iterations?, SIAM J. Matrix Anal. Applic., 13(1992), 778-795.
12. C. T. Kelley: Iterative Methods for Linear and Nonlinear Equations, SIAM, 1995.
13. A. Greenbaum: Iterative Methods for Solving Linear Systems, SIAM, 1997.
14. L. N. Threfethen: Pseudospectra of linear operators, SIAM Review, 39(1997), 383–406.
15. S.V. Patankar, *Numerical Heat Transfer and Fluid Flow*, Hemisphere Publ. Corp.
16. W.E. Pracht, *Calculating Three–Dimensional Fluid Flows at All Speeds with an Eulerian–Lagrangian Computing Mesh*, J. of Comp. Physics, Vol.17, 1975.
17. Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Pub., Boston, 1996.