29th International Multidimensional Engine Modeling User's Group Meeting at the SAE Congress
Detroit, MI, 8 April 2019

Robust preconditioning techniques for iterative solvers in scalable engine simulations using FRESCO

F. Perini[1,2], S. Bnà[3], E. Pascolo[3], I. Spisso[3], R.D. Reitz[1,2]

[1]University of Wisconsin-Madison, USA
[2]Wisconsin Engine Research Consultants LLC, USA
[3]Supercomputing Applications and Innovation (SCAI), CINECA, Italy

Abstract: Advanced iterative solvers and preconditioners were incorporated into the FRESCO engine CFD code. The original solution of linear systems was performed using a matrix-free Conjugate Residual method with Jacobi preconditioning. In this study, a sparse matrix formulation for the Laplacian operator was defined and implemented with the CSR format; a parallel version of the GMRES iterative solver was implemented and tested, along with several LU-decomposition based sparse matrix preconditioners, and a local matrix reordering scheme with the Reverse Cuthill-McKee algorithm. The computational performance of several solver, preconditioner and reordering choices was tested with both a full-mesh and a sector-mesh engine combustion test-cases, in both flow-only configuration and during combustion with hundreds of active species equations. The optimal solver configuration achieved an order of magnitude speedup versus the original setup.

Introduction

Multidimensional computational fluid dynamics (CFD) simulations of engine combustion can support and simplify the design process by providing additional insight that even expensive experimental facilities are not capable of, provided that the right answer is produced in a reasonable amount of time for the combustion engineer. Turnaround times for engine CFD simulations should be well within 10 to 15 hours, i.e., between the time the engineer leaves the office after work and when he/she goes back to it the next day. In order to reduce this time, appropriate usage of High Performance Computing (HPC) resources is crucial: combustion simulations in real-world geometries have complex, large grids which require robust and accurate solvers. Modern users also employ medium-sized reaction mechanisms for combustion kinetics calculations; these can feature hundreds of species, which dramatically increase the number of finite-volume equations being solved, even if they are not all simultaneously active throughout the simulation.

In this work, we addressed linear system solution performance in the FRESCO CFD platform [1], among the most computationally demanding parts of the code while carrying out large-scale simulations. The following solver improvements were addressed:

- *Linear system solver*. The original, KIVA-based matrix-free conjugate residual method [2,3] was replaced by a more efficient, parallel and object-oriented implementation of the Generalized Minimum RESidual (GMRES) solver [4].
- *Matrix handling and preconditioning*. Explicit matrix handling for the Laplacian operators was implemented using the parallel Compressed Sparse Row (CSR) matrix format class [5]. General-purpose, incomplete LU decomposition-based (ILU) preconditioners were implemented and assembled in parallel with the additive Schwarz method.
- *Reordering*. In order to reduce fill-in of the LU decomposition, the bandwidth minimizing symmetric Reverse Cuthill-McKee (RCM) algorithm for matrix reordering was implemented and tested [6];
- *Convergence criteria*. Once the general-purpose GMRES performance bottleneck was identified in its convergence criterion, additional physics-based convergence criteria were defined and evaluated.

An optimal choice of solver and preconditioner parameters was eventually defined for all the equations, and a reduction of solver demand by approximately one order of magnitude was achieved against the original configuration.

The FRESCO platform

This study was performed using FRESCO [1], an object-oriented, parallel platform for multidimensional engine simulations written in modern Fortran. The code implements an unstructured, parallel volume-of-fluid solver for the Navier Stokes equations with automatic domain decomposition for variable-topology meshes. Mesh handling features body-fitted discretization for maximum accuracy. Spray models for fuel injection feature advanced parallel algorithms for breakup, collision, vaporization and near-nozzle flow dynamics [7]. Combustion chemistry is handled by a sparse analytical Jacobian chemistry solver and high-dimensional-clustering based chemistry dimension reduction [5]. The solver implements an explicit first-order time integration scheme using the Arbitrary Lagrangian-Eulerian splitting of Hirt et al. [8], useful for advection-dominated flows such as those in internal combustion engines. First, the Lagrangian derivatives for the momentum (including spray particle coupling), mass conservation, energy and turbulence equations are solved in an implicit

fashion, using a second-order central differencing scheme for the face quantities. Pressure coupling is iterated with the momentum equation using the SIMPLE procedure. Then, the advection terms are computed during a rezoning step by fluxing quantities from the fictitious Lagrangian mesh to the actual, Eulerian node positions using an upwind scheme with van Leer's min-mod flux limiter [9]. An overview of FRESCO's capabilities and research being carried out with it is reported in Figure 1; for a more detailed description of FRESCO and its sub-models, the reader is referred to reference [1].

Linear system solution

First-order implicit time differencing is employed during the moving-with-the-fluid Lagrangian stage of the ALE procedure for several field quantities, in order to improve solution accuracy and allow for faster turnaround due to larger time-steps. For a typical simulation, one must solve one momentum conservation equation, two energy equations (temperature and pressure within the SIMPLE loop), two turbulence equations (i.e. for a RANS k-epsilon model), and a variable number of mass conservation equations per timestep. The following equations were addressed [2]:

Pressure

$$\left[\frac{\partial V}{\partial p}\mathbf{I} - \Delta t \int_V \nabla^2(.)dV\right] p^B = V - V^B + \frac{\partial V}{\partial p}\mathrm{p}_A + \Delta t \sum_f (\mathrm{u}\cdot \boldsymbol{A_f})$$

Temperature

$$\left[\mathbf{I} - \frac{c_{v-term}}{m^B \partial u/\partial T\big|_p}\frac{\Delta t}{Pr_t}\int_V \nabla^2 \mu_t c_p(.)dV\right] T^B = c_{v-term}\left[\tilde{T} + \Delta\tilde{T}_{expl}\right]$$

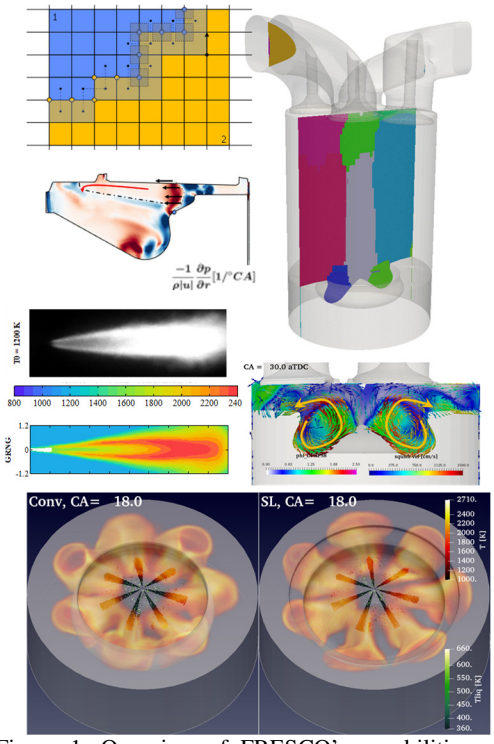

Figure 1. Overview of FRESCO's capabilities and case studies.

where

$$c_{v-term} = \frac{p + p_A}{2}\frac{\partial v/\partial T\big|_p}{m^B\, \partial u/\partial T\big|_p}$$

$$\Delta\tilde{T}_{expl} = \frac{1}{\partial u/\partial T\big|_p}\left\{\frac{p + p_A}{2m^B}\left[\partial v/\partial T\big|_p \tilde{\mathrm{T}} - (\tilde{V} - V_n)\right] + \Delta t\, \boldsymbol{\sigma}(\mathbf{u^B}):\nabla\mathbf{u^B}\right\}$$

Turbulence

The two-equation GRNG k-epsilon model is employed in the current study, solving for turbulence kinetic energy and its dissipation rate:

$$\left[(1+\delta_V)\frac{2}{3}\frac{V_B - V}{V_B} + \frac{\Delta t\, \varepsilon_n}{k_n}\right)\boldsymbol{I} - \frac{\Delta t}{m^B Pr_k}\int_V \nabla^2\mu_t(.)dV\Bigg] k^B = (1-\delta_V)\frac{2}{3}\frac{V_B - V}{V_B}k_n + \Delta t\, \boldsymbol{\sigma}(\mathbf{u^B}):\nabla\mathbf{u^B}$$

$$\left[\left(1 + \delta_V C_{3,G}\frac{V^B - V}{V^B} + C_{2,G}\Delta t\,\frac{\varepsilon_n}{k_n}\right)\boldsymbol{I} - \frac{\Delta t}{m^B Pr_k}\int_V \nabla^2\mu_t(.)dV\right]\varepsilon^B = \frac{\varepsilon_n}{k_n}\frac{\Delta t}{\rho_n}\left[C_{\varepsilon 1}(\boldsymbol{\sigma}_G:\nabla\boldsymbol{u}) - (\boldsymbol{\sigma}:\nabla\boldsymbol{u})\frac{\eta(1-\eta/\eta_0)}{1+\beta\eta^3}\right] - C_{3,G}\frac{V^B - V}{V^B}(1-\delta_V)\varepsilon_n$$

where $\delta_V = V_B > V\,?\; 1 : 0$ is a flag which controls usage of the explicit or implicit value of k and ε during the solution: the implicit value is used whenever the cell is expanding ($V_B > V$), while the time-n value, moved to the right-hand-side of the equation, is used otherwise.

Finally, the momentum equation was not included in the present study: it converges in a few iterations also matrix-free and non-preconditioned [10], and building and preconditioning its three matrices for the *x,y,z* components would certainly make its solution more computationally expensive.

Laplacian matrix formulation

In FRESCO, Laplacian terms for a generic scalar cell field ϕ are approximated employing the divergence theorem to replace the finite-volume integral with a surface integral:

$$\int_V \nabla^2\phi\, dV = \int_V \nabla\cdot(\nabla\phi)\, dV = \int_S \nabla\phi_f \cdot \mathbf{n}\, dS.$$

Face-centered dot products between the field gradient and the face normal are estimated by building a local coordinate system as represented in Figure 2, and changing the basis from the Cartesian coordinate system *(x,y,z)* to this local system. Following the example from [2], the local coordinate system is built connecting the cell centroids from the two cells neighboring the face, as well as two pairs of the face's opposite edge centroids. In case the face is a triangle, one edge will be missing; it is replaced with the face centroid location which, as long as the face has positive area, will still ensure that all three vectors are linearly independent.

The change-of-basis matrix is hence given by:

$$\mathbf{M} = \begin{bmatrix}\mathbf{m_1}\\ \mathbf{m_2}\\ \mathbf{m_3}\end{bmatrix} = \begin{bmatrix} x_{c,nbr} - x_c & y_{c,nbr} - y_c & z_{c,nbr} - z_c \\ x_{e3} - x_{e1} & y_{e3} - y_{e1} & z_{e3} - z_{e1} \\ x_{e2} - x_{e4} & y_{e2} - y_{e4} & z_{e2} - z_{e4}\end{bmatrix};$$

The local face area normal vector can be estimated in the local coordinate system by applying a coordinate change of

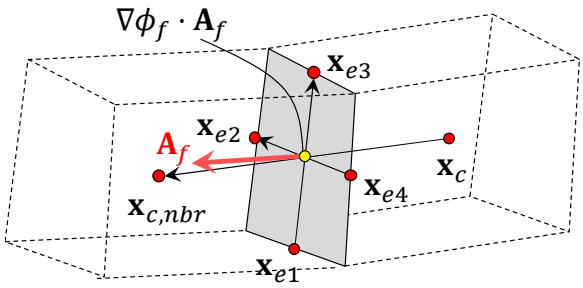

Figure 2. Local coordinate system for face gradient evaluation.

basis, i.e.:

$$\mathbf{M^{-1}A_f}\big|_M = \mathbf{M^{-1}c} = \boldsymbol{A_f}$$

since this local face normal term only depends on the mesh geometry, and it can be pre-computed by solving one 3x3 linear system per face and stored as "geometric coefficients" $\boldsymbol{c}$ which will be a building block of the Laplacian matrix for any scalar fields.

The field gradient term from the local coordinate system can be simply evaluated as:

$$\nabla\phi_f\big|_M = \frac{\partial\phi}{\partial m_1}\boldsymbol{m_1} + \frac{\partial\phi}{\partial m_2}\boldsymbol{m_2} + \frac{\partial\phi}{\partial m_3}\boldsymbol{m_3} = \begin{bmatrix}\phi_{nbr} - \phi_c \\ \phi_{e3} - \phi_{e1} \\ \phi_{e2} - \phi_{e4}\end{bmatrix},$$

i.e., the linear dependency of the local cell Laplacian on neighbor cell values is achieved:

$$\nabla\phi_f \cdot \boldsymbol{A}_f = c_1[\phi_{nbr} - \phi_c] + c_2[\phi_{e3} - \phi_{e1}\,] + c_3[\phi_{e2} - \phi_{e4}],$$

since each edge-centered value is evaluated as the average of its neighbor-cell values:

$$\phi_e = \frac{1}{n_{nbr}}\sum_{i=1}^{n_{nbr}}\phi_{nbr,i}.$$

Each cell's Laplacian eventually depends on a set of face-valued gradient terms which linearly depend on the neighbor cell field values via sums of geometric coefficients (with appropriate sign). So, it is possible to extract a matrix-based Laplacian operator of the scalar cell field:

$$\int_V \nabla^2\phi \, dV = \mathbf{L}\,\phi,$$

$$\mathbf{L} = \int_S \nabla(.) \cdot \mathbf{n} \, dS.$$

The Laplacian matrix $\mathbf{L}$ has (n_{cells} *x* n_{cells}) size and sparse connectivity and can be used to solve systems of equations involving the Laplacian operator. Figures 3 and 4 represent the Laplacian matrix structure for the Sandia 1.9L light-duty single-cylinder engine mesh, with 725k cells, employed in previous studies (e.g., [11]). The entire mesh ("global matrix") was decomposed in 72 ranks using ParMETIS [12], and globally owns a block-diagonal structure. Each CPU only stores its own block ("local matrix"), which is fully sparse, such as represented in Figure 4.

Matrix preconditioning and reordering

All Krylov subspace methods such as GMRES [4] are guaranteed to converge in at most *n* iterations, where *n* is the problem size, i.e., when the Krylov subspace is complete. However, this is usually impractical since *n* - equal to the number of cells in the finite-volume domain – can be a very large number. Preconditioning the system matrix means applying an approximate inverse to the problem, in order to pack the matrix's eigenvalues as close as possible to unity, thus reducing number of iterations needed to achieve convergence. Full LU decomposition preconditioning would lead to the exact matrix inverse, hence solution would be achieved in just one iterations. Hence, 'good' preconditioning is the one which achieves optimal performance as the best trade-off between increasing preconditioning time and decreasing number of solver iterations. In the current GMRES implementation, a preconditioned residual is sought for by solving for a preconditioned residual vector:

$$\mathbf{M}r = b - \mathbf{A}x,$$

where $\mathbf{M}$ is the preconditioning matrix, pre-computed and already stored in terms of an LU decomposition.

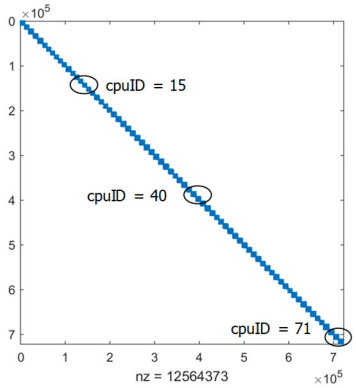

Figure 3. Laplacian sparsity pattern for the Sandia 1.9L engine mesh (725k cells), partitioned in 72 CPUs using ParMETIS [12].

In general, one wants to have a preconditioner which is as close as possible to the full LU decomposition, but with maximum sparsity. As Figure 5 shows, full LU decomposition leads to severe fill-in, i.e., the decomposed matrix is much more dense than the original sparse matrix. This is usually unacceptable both memory- and CPU-intensiveness-wise. Two incomplete LU-decomposition preconditioners were implemented in this study, besides the original matrix-free method:

Jacobi. Jacobi preconditioning, or the original option, is a successful choice because of its limited, *O(n)*, memory and evaluation requirements; however, this is the simplest and possibly less accurate preconditioner choice. It essentially just scales the diagonal elements, while discarding all connectivity-based information of the matrix.

ILU0. The simplest form of incomplete LU preconditioner

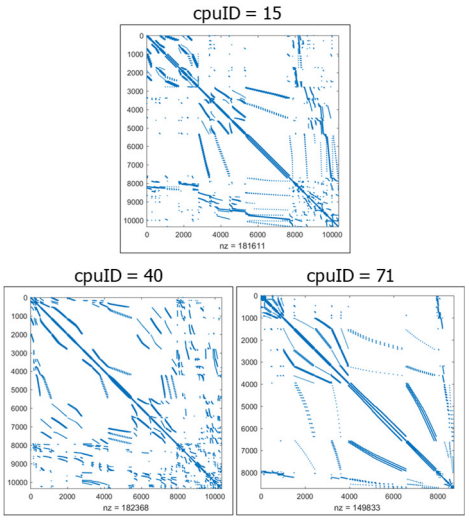

Figure 4. Sparsity patterns of local Laplacians of CPUs 15, 40, 71.

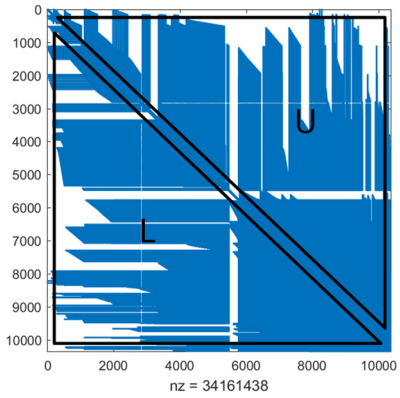

Figure 5. Sparsity pattern of the full LU decomposition for the Laplacian matrix block of rank 40.

assumes that the sparsity structure of the LU decomposition is the same as the non-inverted matrix sparsity. This strategy is memory-efficient as structure reallocations are avoided; though a pretty crude approximation, as ILU's structure is completely unrelated with the actual LU structure, it's still more complex than the original diagonal preconditioning, and often used as a good non-optimized preconditioner choice [13].

ILUT. Saad's incomplete LU with dual truncation strategy was implemented as well [14]. It performs a flexible truncated sparse LU decomposition based on two strategies: the sparsity structure order *(k*, or level-of-fill*)* is truncated to the sparsity structure of A^k; and further off-diagonal elements are dropped if they're smaller – by a threshold, ε_d – compared to their corresponding diagonal value.

Reordering. We implemented the symmetric RCM matrix reordering algorithm for *local* matrix reordering, with the aim of improving preconditioner performance. Global ordering is still performed by the ParMETIS domain decomposition algorithm; in this way, expensive MPI communications are avoided, and solver-optimal matrix ordering can be achieved at the local level. Figure 6 shows

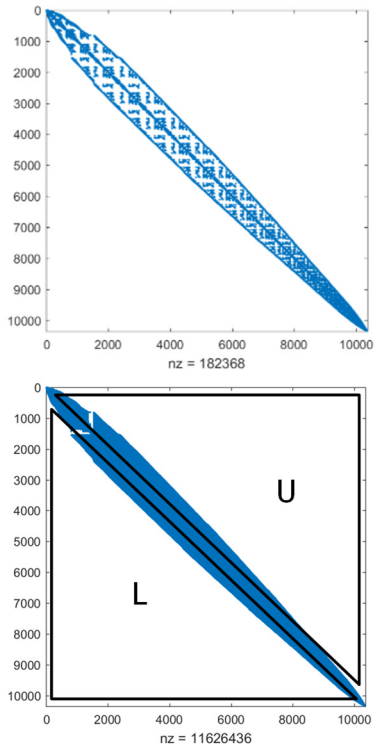

Figure 6. (top) Sparsity pattern of the rank 40 Laplacian matrix, reordered using the RCM algorithm; (bottom) full LU decomposition of the reordered matrix.

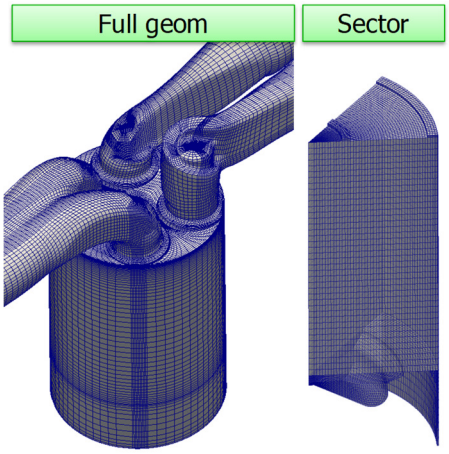

Figure 7. View of the full and sector meshes employed in this study.

mesh	n_{cells}	n_{verts}
sector	*99641*	*104943*
full	*724055*	*753735*

Table 1. Test-case mesh properties.

the effects of matrix ordering on the local matrix of cpu rank #40: the ordered matrix has a much smaller bandwidth, and its full LU decomposition has 65.4% fewer non-zero elements than the non-sorted matrix's one. Because LU fill-in now happens only within the original matrix bandwidth, also the ILU preconditioners can also benefit from

reordering: the ILU structure is not unrelated to the original matrix's structure anymore, and even the ILU0 preconditioner can provide a decent approximation of the decomposition.

Convergence criterion. The GMRES method based on residual norm minimization. During this inner iterative procedure, a residual norm is always known, hence, its convergence check compares the norm of the residual with its initial value [4]. We found this condition to bee too restrictive for our case, where each solution is initialized with a good approximation, extrapolated from the previous step. So, physics-based convergence criteria were implemented based on those of [2]. These checks can only be performed once per iteration, while the original initial-residual criterion is kept within the least squares iteration.

Results and discussion

We tested several solver and preconditioner configurations against two reference engine simulation setups [11], modeling the Sandia 1.9L optical platform, represented in Figure 7: one full engine geometry, which also includes ports, runners and intake/exhaust plenums, and one 1/7th cylinder sector mesh case. All simulations were run on Cineca's Galileo supercomputer, each compute node equipped with 2x18-core Intel Xeon E5-2697v4 CPUs and 128GB RAM. For both meshes, a full IVC to EVO simulation was run, and a 100-timestep test region was selected out of the full time range, where both spray and combustion were present.

Sector simulations. As reported in Figure 8, the pressure equation dominates over the whole number of linear solver iterations, regardless of the solution approach. The Jacobi-preconditioner-backed conjugate residual solver, as well as GMRES with the non-reordered ILU0 preconditioner, needed to converge more than one order of magnitude more iterations than all other configurations. As expected, the ILU0 preconditioner benefited the most from matrix reordering (RCM employed), but reordering was

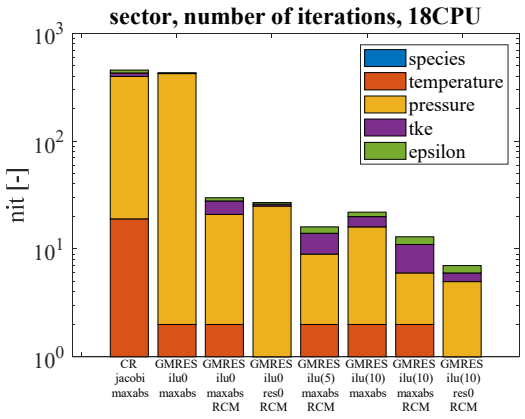

Figure 8. Sector mesh testcase, number of solver iterations per equation, 18CPU domain decomposition.

nevertheless beneficial for all solver configurations, in particular for the badly conditioned pressure equation.

As Figure 9 summarizes, the cost of building the preconditioner is reduced by a factor between 3 to 5 when employing matrix reordering, thanks to the reduced fill-in and reduced cache misses due to smaller bandwidth; there is up to one order of magnitude increase in wall-time for improved accuracy moving from ILU0 to ILU(10). In all cases, these relative relationships do not change as the matrix size is reduced, by increasing the number of CPUs. The cumulative cost of the iterations is significantly affected by the preconditioner choice, ILU(10)+RCM performing the best for all numbers of CPUs tested.

Full mesh. The full geometry tests were conducted well into combustion, where the flow non-uniformities are relevant. Figure 10 shows, solver performance is dramatically affected by the preconditioner choice. The conjugate residual solver with Jacobi preconditioning required a total of 180 solver iterations, versus a minimum of 4 for GMRES with ILU(10) preconditioner and initial residual-based convergence. GMRES with ILU(10) and physics-based convergence exhibited second-best performance with 13 total solver iterations.

Finally, computational performance in Figure 11 shows the amount of time being spent on building the preconditioner, solving for the inverse of the preconditioned matrix into a $M^{-1} \cdot v$ matrix-vector product, and other iteration time such as for evaluating the linear system's residual vector, or right hand side.

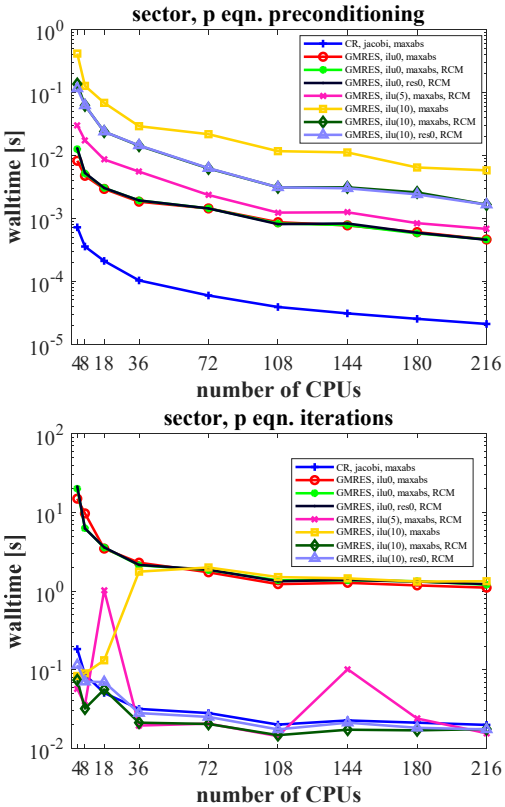

Figure 9. Cost of (top) building the preconditioner and (bottom) solving the linear system vs. number of CPUs for the sector geometry.

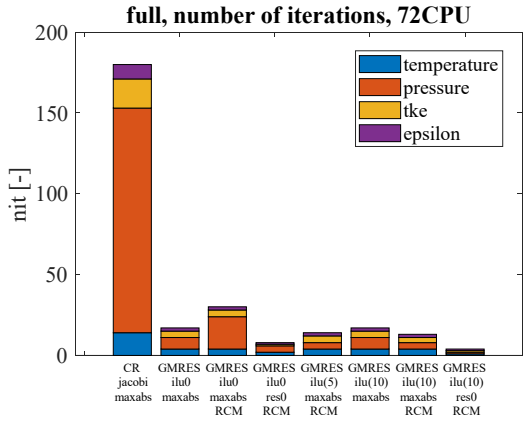

Figure 10. Full mesh testcase: number of solver iterations per equation, 72CPU domain decomposition.

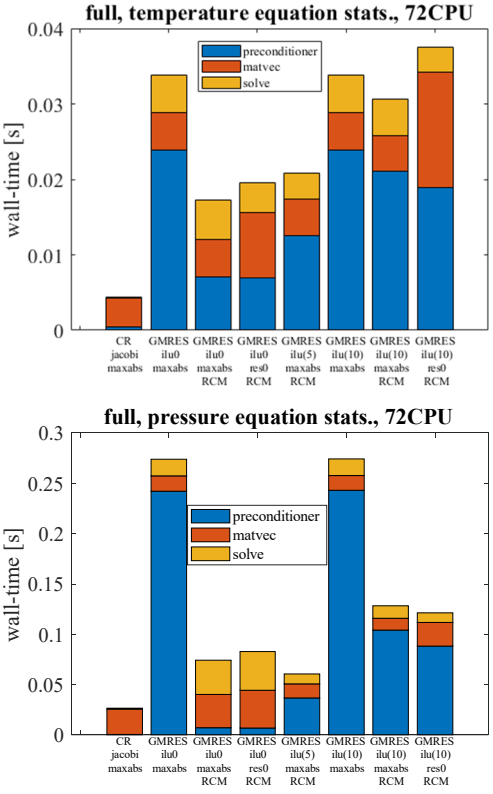

Figure 11. Linear solver performance profiling for (top) temperature and (bottom) pressure equations. 'preconditioner' indicates building time; 'matvec' indicates preconditioned matrix-vector products; 'solve' indicates residual and right-hand-side calculation.

Conclusions

We implemented advanced matrix handling, preconditioners and solvers for multidimensional engine simulations, including improvements with reordering and solver convergence criteria. Performance tests against full-geometry and sector configurations highlighted an optimal configuration made of the GMRES solver with a locally-reordered matrix and an ILU(10) preconditioner, independent of the number of CPUs used. The optimal configuration achieved a reduction in number of solver iterations by more than one order of magnitude against the original code setup. Future work fill focus on run-time optimization of preconditioner and solver settings for maximum speedup.

Acknowledgements

The authors gratefully acknowledge support and computational resources for this work at the Cineca super-computing center by the EC Research Innovation Action under the H2020 Program, through HPC-EUROPA3 grant HPC17FFPU6.

References

1. Perini, F., Reitz, R.D., "FRESCO – an object-oriented, parallel platform for internal combustion engine simulations", International Multidimensional Engine Modeling User's Group Meeting at the SAE Congress, 2018.
2. Torres D.J., Trujillo M.F., "KIVA-4: an unstructured ALE code for compressible gas flow with sprays", Journal of Computational Physics 219(2), 943-975, 2006.
3. M.J. Holst, "Notes on the KIVA-II Software and Chemically Reactive Fluid Mechanics", Lawrence Livermore National Laboratory, URCL-ID-112019, 1992.
4. Saad, Y., Schultz, M.H., "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems", SIAM J. Sci. and Stat. Comput. 7(3), 856-869, 1986.
5. Perini F., Galligani E., Reitz R.D., "An analytical Jacobian approach to sparse reaction kinetics for computationally efficient combustion modelling with large reaction mechanisms", Energy&Fuels 26 (8), 4804-4822, 2012.
6. Liu, W.-H., Sherman, A.H., "Comparative Analysis of the Cuthill-McKee and the Reverse Cuthill-McKee Ordering Algorithms for Sparse Matrices", SIAM J. Numer. Anal., 13(2), 198-213, 1976.
7. F. Perini and R.D. Reitz, "Improved atomization, collision and sub-grid scale momentum coupling models for transient vaporizing engine sprays", International Journal of Multiphase Flows 79(2016), 107-123.
8. Hirt C.W., Amsden A.A., Cook J.L., "An arbitrary Lagrangian-Eulerian computing method for all flow speeds", Journal of Computational Physics 14(3), 227-253, 1974.
9. Van Leer B., "Towards the ultimate conservative difference scheme II. Monotonicity and conservation combined in a second order scheme", Journal of Computational Physics 14 (4): 361–370, 1974.
10. L. Arnone, P. D'Ambra, S. Filippone, "A Parallel Version of KIVA-3 based on General Purpose Numerical Software and its Use in Two-Stroke Engine Applications", Practical Parallel Computing, ISBN 1-59033-127-3, 2001.
11. Perini F., Busch S., Kurtz E., Warey A., Peterson R.C., Reitz R.D., "Limitations of Sector Mesh Geometry and Initial Conditions to Model Flow and Mixture Formation in Direct-Injection Diesel Engines", SAE Technical Paper 2019-01-0204, 2019.
12. Karypis G., Kumar V., "A fast and high quality multilevel scheme for partitioning irregular graphs", SIAM Journal on Scientific Computing 20(1), 359-392, 1998.
13. Balay, S., Abhyankar, S., Adams, M., Brown, J., Brune, P., Buschelman, K., Dalcin, L. D., Eijkhout, V., Gropp, W., Kaushik, D., Knepley, M., May, D., McInnes, L. Curfman, Munson, T., Rupp, K., Sanan, P., Smith, B., Zampini, S., Zhang, H., and Zhang, H.. PETSc Users Manual Revision 3.8. United States: N. p., 2017. Web.
14. Y. Saad, "ILUT: a dual threshold incomplete LU factorization", in: Numerical Linear Algebra With Applications, Wiley, 1994.