International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2011)
Rio de Janeiro, RJ, Brazil, May 8-12, 2011, on CD-ROM, Latin American Section (LAS) / American Nuclear Society (ANS)
ISBN 978-85-63688-00-2

PARALLELIZED PRECONDITIONED BICGSTAB SOLUTION OF SPARSE LINEAR SYSTEM EQUATIONS IN F-COBRA-TF

René van Geemert, Markus Glück[*] , Michael Riedmann, Harry Gabriel
AREVA NP GmbH
Wetterkreuz 15, D-91058 Erlangen (Germany)
rene.vangeemert@areva.com, michael.riedmann@areva.com, harry.gabriel@areva.com

ABSTRACT

Recently, the in-house development of a preconditioned and parallelized BiCGStab solver has been pursued successfully in AREVA's advanced sub-channel code F-COBRA-TF. This solver can be run either in a sequential computation mode on a single CPU, or in a parallel computation mode on multiple parallel CPUs. The developed procedure enables the computation of several thousands of successive sparse linear system solutions in F-COBRA-TF with acceptable wall clock run times. The current paper provides general information about F-COBRA-TF in terms of modeling capabilities and application areas, and points out where the relevance arises for the efficient iterative solution of sparse linear systems. Furthermore, the preconditioning and parallelization strategies in the developed BiCGStab iterative solution approach are discussed. The paper is concluded with a number of verification examples.

Key Words: two-phase flow, sub-channel modeling, sparse linear systems, BiCGStab, preconditioning, parallelization

1. INTRODUCTION

F-COBRA-TF is AREVA's advanced sub-channel code for physical modeling of two-phase flow in both PWRs and BWRs (cf. Figure 1). It is based on a heterogeneous two-fluid representation of the two-phase flow in the reactor core, solving the transport equations for mass, momentum, and energy for the three separate fields continuous liquid, vapor, and entrained liquid (droplets in the vapor core) in a transient way. Different flow regimes are distinguished for taking into account phenomena like wall and interfacial drag, wall and interfacial heat transfer, turbulent mixing, void drift, entrainment, deposition, etc. by means of continuously improved physical models. By means of a mechanistic individual film model each liquid film on any rod or unheated structural surface within a sub-channel is treated separately. This feature enables an improved prediction of dry-out [1].

The code has been validated extensively, by now primarily for typical steady-state and transient BWR conditions with respect to single- and two-phase pressure losses [2], void distribution [3], and turbulent mixing [1]. This comprehensive validation has been performed against well-known experiments from open literature, tests within the framework of the OECD/NRC BFBT benchmark, and AREVA in-house experiments carried out at ATRIUM^{TM} 10 fuel assemblies.

F-COBRA-TF is currently used for:

[*] Current working address: University of Applied Sciences Jena, Carl-Zeiss-Promenade 2, 07745 Jena (Germany) , E-Mail: markus.glueck@fh-jena.de

- a wide variety of design studies (e.g., within the development phase of AREVA's next generation BWR fuel assembly ATRIUM™ 11)
- measurement decisions
- reference calculations
- product decisions

as depicted in Figure 2. From the physical point of view, the application areas of F-COBRA-TF can be classified according to the illustration in Figure 3.

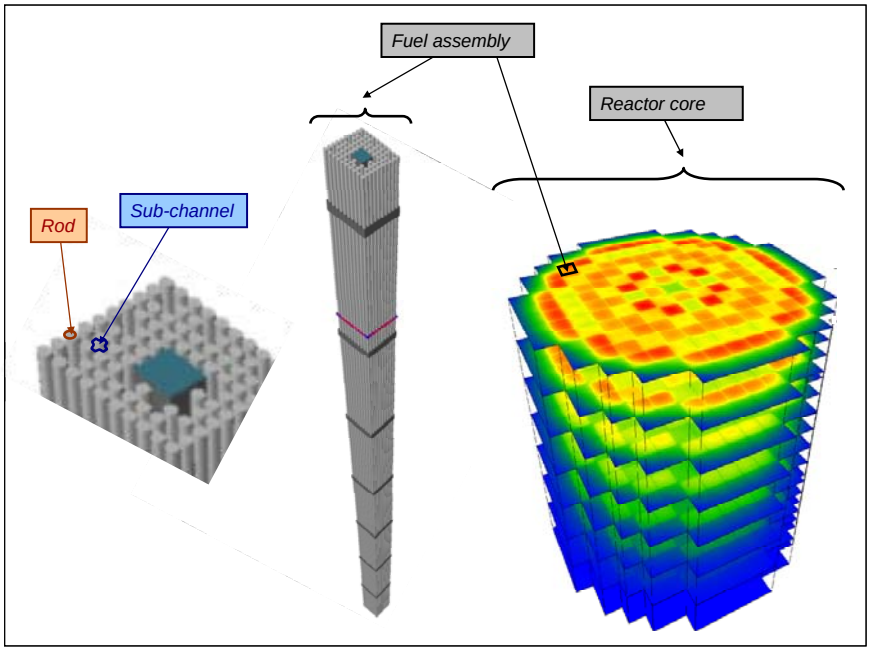

Figure 1. Schematic overview of the geometrical modeling capability range of F-COBRA-TF.

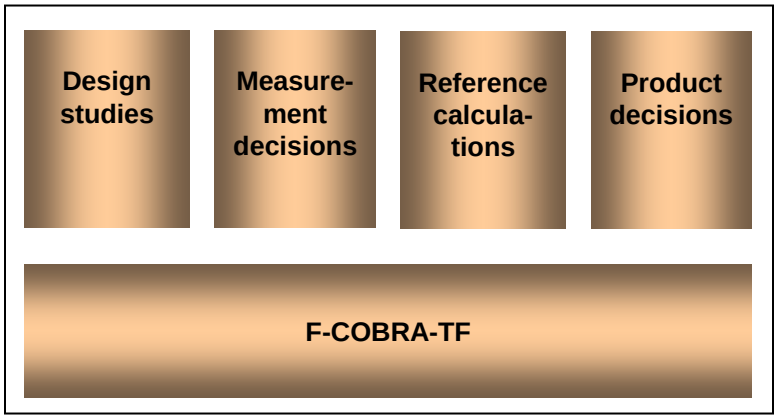

Figure 2. Schematic overview of application areas of F-COBRA-TF.

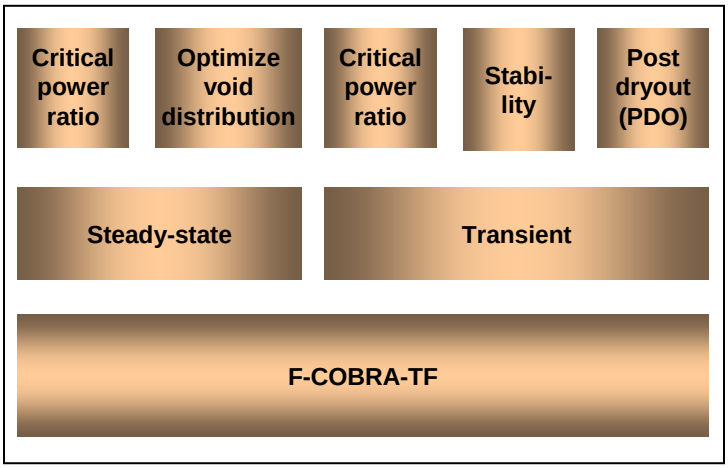

Figure 3. Schematic overview of physical modeling capabilities included in F-COBRA-TF.

1.1. Basic equations in F-COBRA-TF

A schematic overview over the basic transport equations solved in F-COBRA-TF is shown in Figure 4:

- Each field (continuous liquid, entrained liquid, and vapor) has its own mass equation. In addition, there is a mass equation for the mixture of non-condensable gases which might occur beside the water-vapor mixture.
- Each field has its own axial momentum equations and lateral momentum equations, whereas the momentum equations for vapor include also the non-condensable gas mixture. The background of this simplification is the assumption that the gases move with approximately the same velocity as the vapor.
  Note: There is the specialty in sub-channel formulation that the lateral momentum equation represents not only one but two Cartesian coordinate directions. That means each velocity component perpendicular to the axial direction and thus acting in a horizontal layer is a lateral velocity.
- Each phase (liquid and vapor) has its own energy equation. The liquid energy equation contains both fields continuous liquid and entrained liquid for the sake of simplicity. That means both liquid fields are assumed to have the same enthalpy / temperature. However, there are many calculation procedures that take into account the difference between these two fields. E.g., if it is the question how to partition the evaporation of liquid, i.e., which fraction evaporates from the continuous liquid film and which amount evaporates from the entrained droplets. Furthermore, the vapor energy equation includes also the non-condensable gas mixture, based on the assumption that the gas mixture has the same temperature as the vapor.

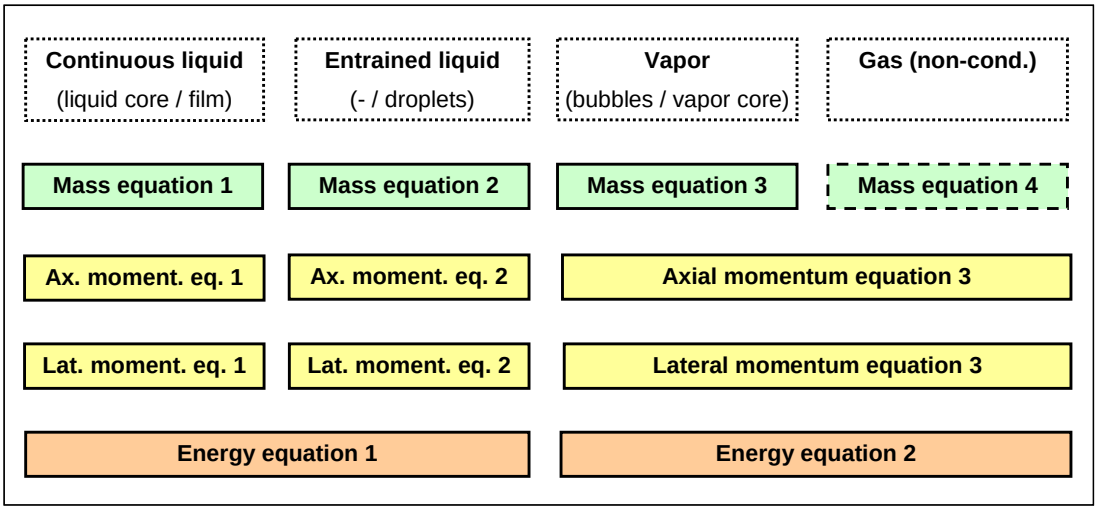

Figure 4. Conservation equations in F-COBRA-TF (physically classified)

1.2. Characterization of the computational mesh

F-COBRA-TF is based on a so-called *staggered grid* to solve the fluid mechanical problem, whereas different variables are solved and stored at different (staggered) positions in the computational domain. As shown in Figure 5, the scalar variables are always saved in the center of scalar mesh cells, whereas the axial mass flow rates are saved at the boundaries between the scalar mesh cells and thus – in case of a non-equidistant grid – *not* in the center of the axial momentum mesh cells.

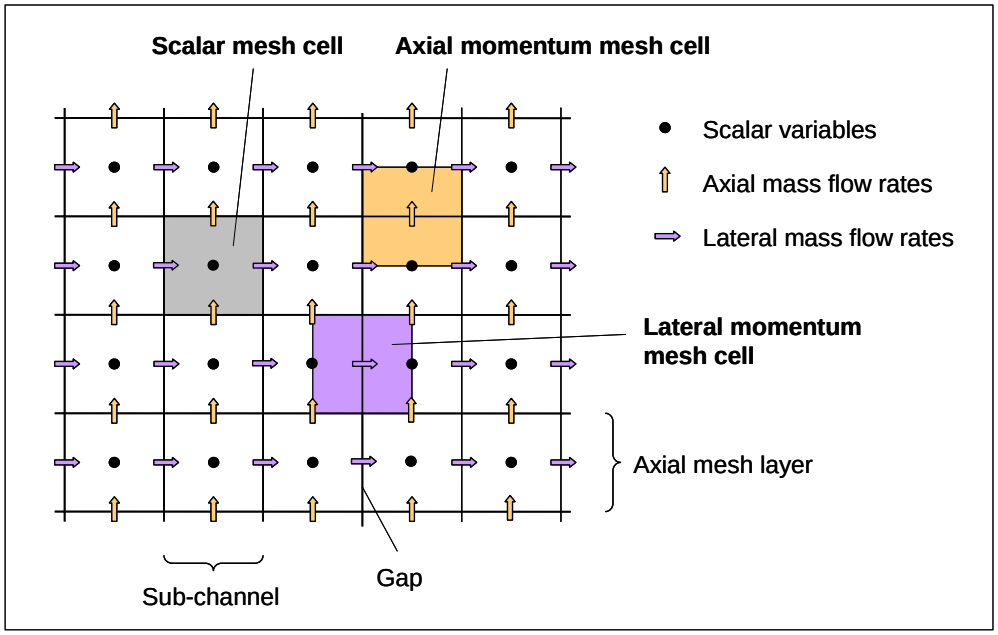

Figure 5. Schematic of staggered **variable and control volume arrangement in F-COBRA-TF (in a vertical cutting plane through the computational domain)**

1.3. Numerical solution approach for solving the conservation equations

The basic conservation equations are solved by means of a semi-implicit method. Specifically this means that:

- the axial and lateral momentum equations are solved *implicitly* with respect to the coupling between the three fields continuous liquid, entrained liquid, and vapor. In each cell the mass flow rates for these three fields are solved simultaneously.

  However, the momentum equations are solved *explicitly* in the sense that all spatial dependencies from neighbor cells are based on values known from the old time step.

- the mass and energy equations are solved *implicitly*, that means they are solved simultaneously for all scalar fluid mesh cells of the computational domain.

An overview of the entire numerical sequence to solve the conservation equations is illustrated in 6. The efficient solution of the blue marked step *'Solution of system pressure matrix'* is the main topic of the current paper and will be discussed in detail in the following sections.

2. ITERATIVE SOLUTION OF THE SPARSE LINEAR PRESSURE EQUATIONS

The solution of the system pressure matrix boils down to the iterative solution (using the BiCGStab algorithm [4]) of large sparse, non-symmetric linear systems. These linear equations have the form

$$\left[\mathbf{1} - \mathbf{A_{LD}} - \mathbf{A_{UD}} \right] \mathbf{x} = \mathbf{b} \tag{0.1}$$

with **1** denoting the unity matrix and $\mathbf{A_{LD}}$ and $\mathbf{A_{UD}}$ denoting the lower and upper diagonal parts of the coupling matrix **A**. The latter quantifies the physical coupling between neighboring channels. If possible, it is always recommendable to impose a *lower diagonal preconditioning (LDP) measure* (i.e. Gauss-Seidel preconditioning) that leads to the following preconditioned system:

$$\left[\mathbf{1} - (\mathbf{1}-\mathbf{A}_{\mathrm{LD}})^{-1} \mathbf{A}_{\mathrm{UD}} \right] \mathrm{x} = \tilde{\mathrm{b}} \tag{0.2}$$

with the transformed source

$$\tilde{\mathbf{b}} = (\mathbf{1}-\mathbf{A_{LD}})^{-1} \mathbf{b} \tag{0.3}$$

This preconditioning measure typically presents the highest benefit in case of a *red-black partitioning* of the sub-channel grid, if such a partitioning is possible. In any case, the LDP measure presents hardly any additional computational cost and, in the case of a red-black partitioning, can be parallelized consistently; one simply needs to arrange the loop for computing $(1-A_{LD})^{-1} A_{UD}$ in two successive half-grid loops (i.e. the red loop and the black loop).

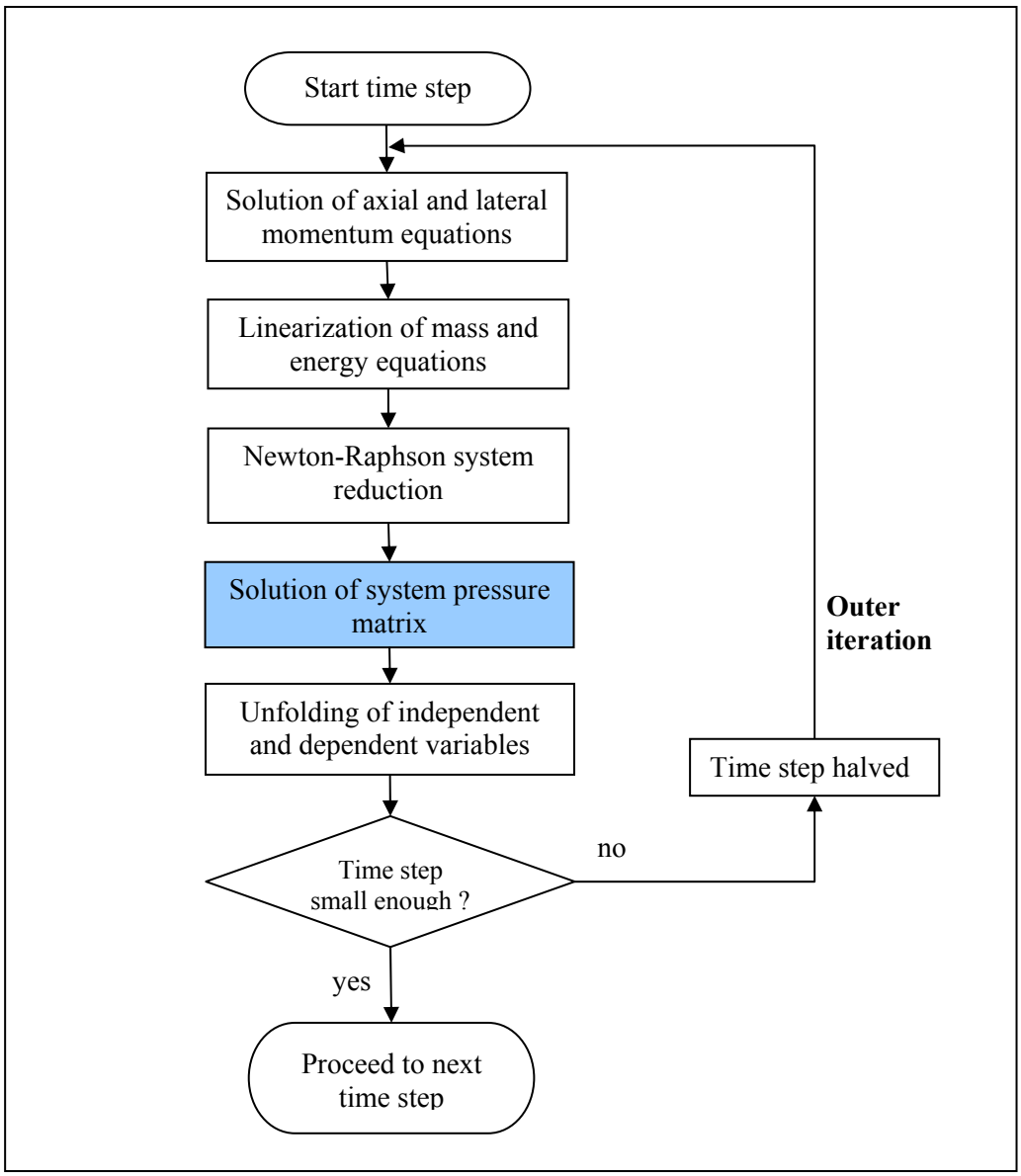

Figure 6. Flow chart of the numerical solution processes for all the relevant conservation equations in F-COBRA-TF.

2.1. Relevance of applying BICGSTAB as a suitable iterative solution algorithm

With the diagonal matrix being always a unity matrix, the condition number of the sparse linear system follows from the norm of the non-diagonal operator. The latter is a measure for how fast any iterative linear system solution process can be expected to converge. In the F-COBRA-TF applications of current interest, this number is typically quite close to 1 (such as 0.995 or higher) even in the case of applying lower diagonal preconditioning (LDP).

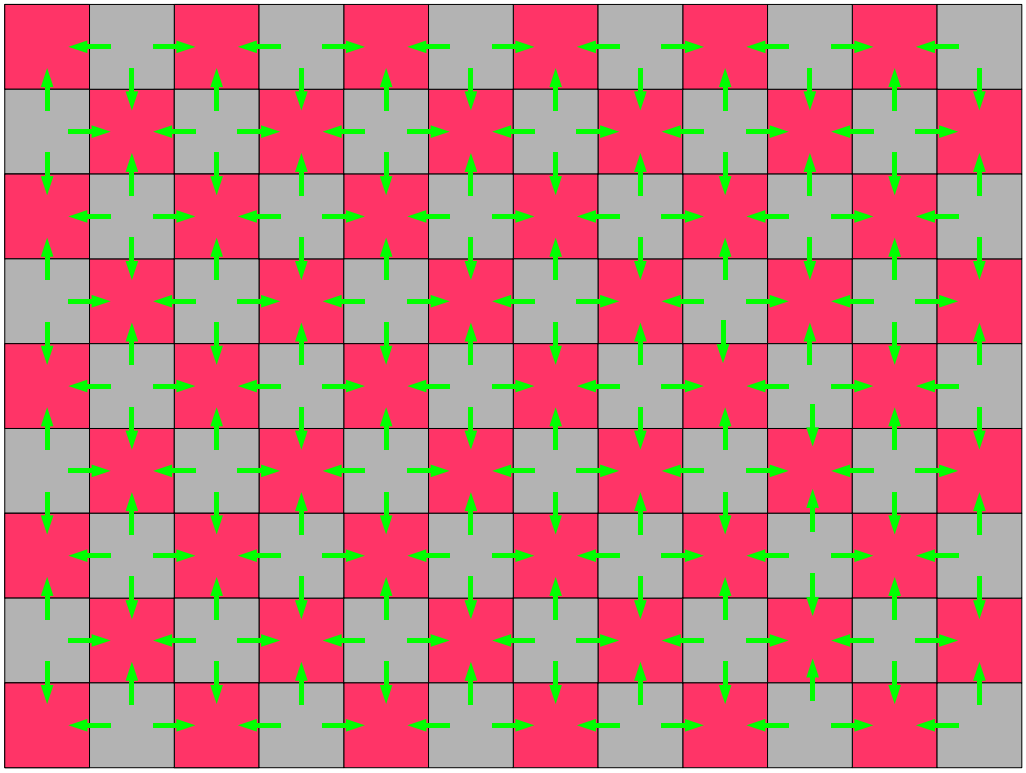

Figure 7. Example of a red-black partitioning of a 2D sub-channel mesh ; each red sub-channel interacts directly only with its directly adjacent, oppositely colored (i.e. black) neighbors ; the associated matrix structure enables optimal numerical benefits of lower diagonal preconditioning.

With condition numbers this high, one should not rely on straightforward untransformed Jacobi iterations, defined by

$$\mathbf{x}^{(n+1)} = \tilde{\mathbf{b}} + (\mathbf{1}-\mathbf{A}_{LD})^{-1}\mathbf{A}_{UD}\,\mathbf{x}^{(n)} \qquad (0.4)$$

for the iterative solution of these equations. This is because the asymptotic convergence error decay ratio in Jacobi iteration is equal to the condition number $\| (1-A_{LD})^{-1}\ A_{UD} \|$ of the non-diagonal matrix (i.e. 0.995 or higher, which means a horribly slow error decay for Jacobi). As an example, achieving a typically necessary convergence error reduction of 10^{-6}, while having an error decay ratio of 0.995, would require about $-6/\log(0.995) \approx 2700$ Jacobi iteration steps and associated matrix-vector multiplications. Under such kinds of conditions (and also without any additional preconditioning measure), the BiCGStab algorithm [4], when compared to the Jacobi algorithm, usually achieves the same convergence error reduction while requiring substantially fewer iterative steps. BiCGStab is not difficult to program and, especially for LDP preconditioned linear systems, it is a clearly more powerful solution algorithm than the Jacobi (i.e. power) method.

2.2. Further preconditioning possibilities

In case of linear systems with condition numbers that are particularly close to 1, a further preconditioning (i.e. beyond the basic LDP measure) of the linear system could prove beneficial. This is due to the property that the convergence rate of BiCGStab (in terms of the necessary numbers of BiCGStab iterations to be pursued) improves more than linearly versus a decrease in condition number of the non-diagonal matrix. The final computational performance is, however, determined not only by the necessary numbers of BiCGStab iterations to be pursued, but also by the computational cost of a single BiCGStab iteration.

The beauty of the basic LDP measure is that it does not raise the necessary number of arithmetic operations per matrix-vector multiplication, i.e. one has an equally lean matrix-vector multiplication kernel. Because of this, one gets the benefit of a lower condition number (and hence a higher convergence rate) without having to pay a computational price for this. Interestingly, a partial degradation in this respect can however occur in parallel computation mode. Here, a computational penalty can arise due to a higher OpenMP overhead imposed by a loop restructuring that can prove necessary for LDP (cf. section 3).

Now, further preconditioning (beyond LDP) basically implies rearranging the LPD-preconditioned linear system such that another linear system is shaped that has a clearly lower condition number (and hence a clearly stronger diagonal dominance) while having exactly the same (and unique) solution as the untransformed system. The price for this is having to accept a less sparse non-diagonal matrix structure so that the matrix-vector multiplication kernel becomes less lean. Thus, whether a final speedup results from further preconditioning measures (beyond basic LDP) depends on how much the condition number can be further reduced vs. how much more expensive the matrix-vector multiplication will get due to the added preconditioning measure.

Nonetheless, with condition numbers that are extremely close to 1, it may yet pay off to use a somewhat more involved preconditioning approach. It is in fact possible, based on a computationally cheap estimation of the system condition number prior to solving the linear system, to have an automated decision on which preconditioning variant to apply. In case of a further preconditioned system, realized for example by

(i) transformation of the linear system equation,
(ii) and including an optimized acceleration parameter ω in the transformation

one can expect to get a transformed (and less sparse) linear system that features a significantly lower condition number. The transformation, including the use of a scalar parameter ω, is arranged as follows:

$$\mathbf{x} - \left[\omega \, (\mathbf{1}-\mathbf{A}_{LD})^{-1} \, \mathbf{A}_{UD} + (\mathbf{1}-\omega)\mathbf{1} \right] \mathbf{x} = \omega \, \tilde{\mathbf{b}} \qquad (0.5)$$

with $1 \leq \omega < 2$. Now, defining the operator Θ_ω as

$$\boldsymbol{\Theta_\omega = \omega\,(1\text{-}A_{LD})^{-1}\,A_{UD} + (1\text{-}\omega)\,1} \tag{0.6}$$

the transformed system boils down to

$$\boldsymbol{\left[1 - \Theta_\omega\right]\ x = \omega\,\tilde{b}} \tag{0.7}$$

According to [5], with the μ_n being the eigenvalues of $(1\text{-}A_{LD})^{-1}\,A_{UD}$ whose absolute values are bounded by μ_0 (i.e. ... $< |\mu_2| < |\mu_1| < |\mu_0|$), the ω-dependence of the operator norm $\|\Theta_\omega\|$ adheres approximately to

$$\boldsymbol{\left\|\Theta_\omega\right\| = \max_{|\mu_n| < \mu_0}\left|\frac{\omega\mu_n}{2} \pm \sqrt{\frac{\omega^2\mu_n}{4} - (\omega\text{-}1)}\right|^2} \tag{0.8}$$

Thus, the specific value of ω that minimizes $\|\Theta_\omega\|$ follows from the analytical mini-max problem

$$\boldsymbol{\left\|\Theta_\omega\right\|_{min} = \min_\omega\left\{\max_{|\mu_n| < \mu_0}\left|\frac{\omega\mu_n}{2} \pm \sqrt{\frac{\omega^2\mu_n}{4} - (\omega\text{-}1)}\right|^2\right\}} \tag{0.9}$$

for which the analytical solution is the known classical formula [5]:

$$\boldsymbol{\omega = \frac{2}{1+\sqrt{1-\mu_0^2}}} \tag{0.10}$$

Choosing ω in this way leads to the following theoretical prediction for the norm $\|\Theta_\omega\|$:

$$\boldsymbol{\left\|\Theta_\omega\right\| = \omega - 1} \tag{0.11}$$

Figure 8 shows the theoretical prediction of $\|\Theta_\omega\|$ as a function of ω for different given values for μ_0. One can see clearly that the norm $\|\Theta_\omega\|$ can be expected to be significantly lower than the norm $\|(1\text{-}A_{LD})^{-1}\,A_{UD}\|$. The higher μ_0 (whose value is typically correlated with grid refinement and grid size), the higher the value of ω for which $\|\Theta_\omega\|$ is minimized, and the more sensitively $\|\Theta_\omega\|$ responds to small variations $\omega + \delta\omega$ around ω.

The direct relationship between ω and μ_0 (i.e. corresponding to Eq.(1.9)) is displayed in Figure 9. But: one needs to further transform Eq.(1.7). This is necessary as Eq.(1.7) is, within a BiCGStab solution process, still fully equivalent to the untransformed, LDP-preconditioned Eq.(1.2). The simple solution to this is to pre-multiply both sides of Eq.(1.7) with $1+\Theta_\omega$, which leads to the following final transformed equation:

$$\left[\mathbf{1} - \boldsymbol{\Theta}_\omega^2\right] \quad \mathbf{x} \quad = \quad \tilde{\tilde{\mathbf{b}}} \tag{0.12}$$

with the following final expression for the transformed source (to be computed once-only prior to the iteration process):

$$\tilde{\tilde{\mathbf{b}}} \quad = \quad \left[\mathbf{1} + \boldsymbol{\Theta}_\omega\right] \quad \boldsymbol{\omega}\,(\mathbf{1}-\mathbf{A}_{\mathbf{LD}})^{-1}\,\mathbf{b} \tag{0.13}$$

The operator Θ_ω^2 is, however, clearly less sparse than the original LDP-preconditioned operator $(1-A_{LD})^{-1}\ A_{UD}$.

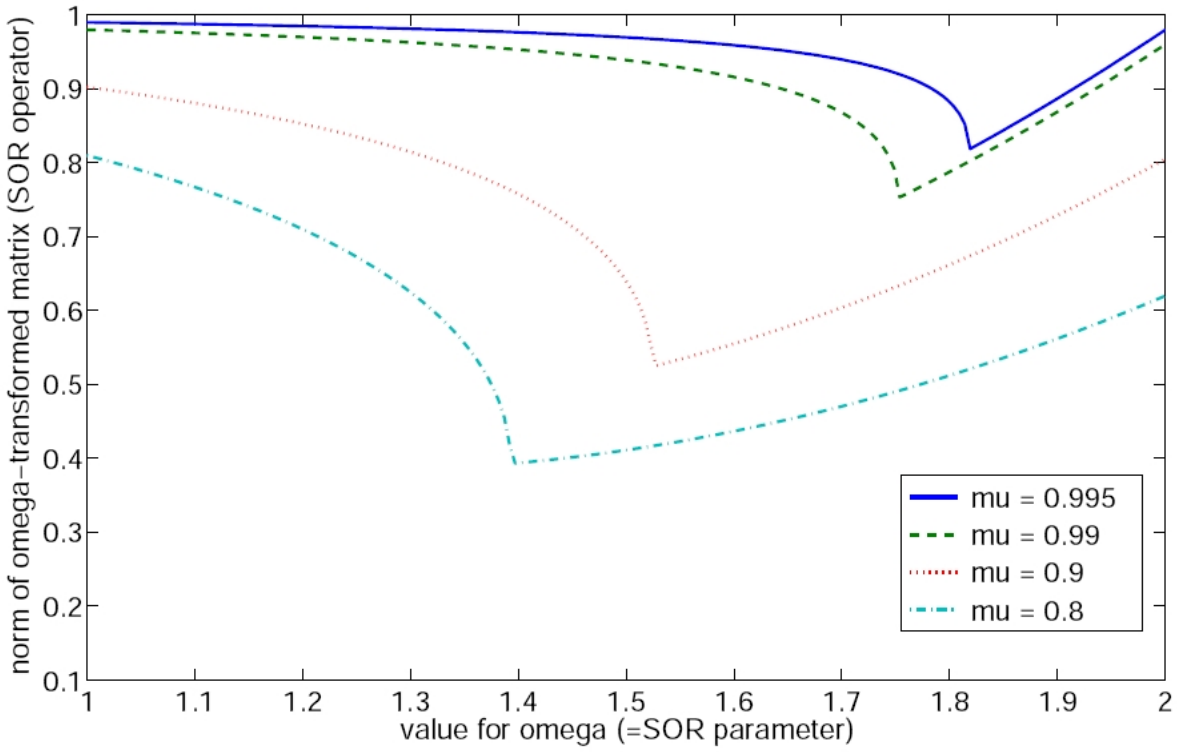

Figure 8. Dependence of $\|\Theta_\omega\|$ on the choice of ω for different values of μ_0.

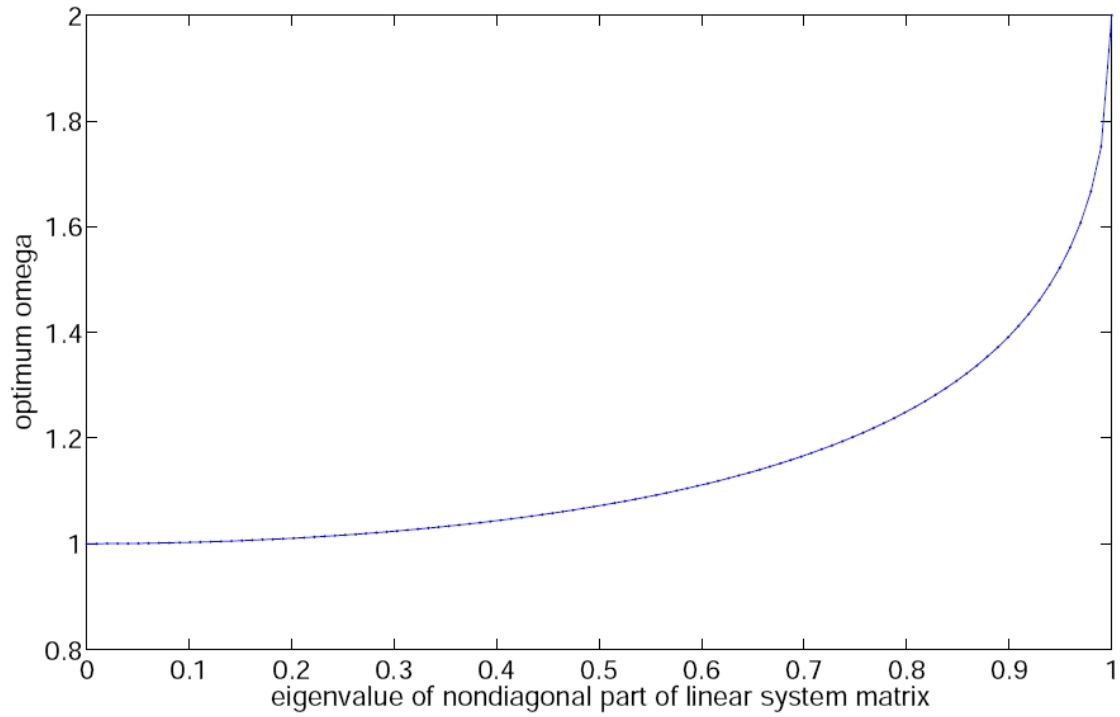

Figure 9. Direct relationship between ω and μ_0 in correspondence with Eq. (1.9)

Within the F-COBRA-TF BiCGStab iteration framework for solving the transformed and parametrized sparse linear system Eq.(1.12), the optimum convergence behavior can be achieved very easily if one has a quick and computationally cheap way of computing μ_0. With this value available, the optimum parameter value ω follows directly and analytically from formula (1.10). This means that, for each of the many successive individual linear system solution processes within a large core cycle or transient computation, one will automatically and always get the optimum error decay rate.

It is not difficult to arrange a computationally cheap estimation of the fundamental eigenvalue μ_0. One can, for example, use flat approximations (i.e unity vectors) for both adjoint and forward fundamental eigenvectors of $(1\text{-}A_{LD})^{-1}\ A_{UD}$. For the Rayleigh quotient approximation of μ_0 this yields a simple and effective formula for the eigenvalue estimate $\bar{\mu}_0$:

$$\bar{\mu}_0 = \frac{< 1 \mid (1\text{-}A_{LD})^{-1}\ A_{UD} \mid 1 >}{< 1 \mid 1 >} \tag{0.14}$$

This formula boils down to

$$\bar{\mu}_0 = \frac{1}{N} \sum_{i=n}^{N} \left[\sum_{i'(i)} \left[(1\text{-}A_{LD})^{-1}\ A_{UD} \right]^{ii'} \right] \tag{0.15}$$

which is the normalized sum over all nonzero elements in the sparse non-diagonal matrix (N is the total number of fuel mesh cells). Inserting this in Eq.(1.9), we get a formula for ω that enables its computation as a direct analytical function of the system matrix $(1\text{-}A_{LD})^{-1}\ A_{UD}$:

$$\omega = \frac{2}{1 + \sqrt{1 - \left(\frac{1}{N} \sum_{i=n}^{N} \left[\sum_{i'(i)} \left[(1\text{-}A_{LD})^{-1}\ A_{UD} \right]^{ii'} \right] \right)^2}} \qquad (0.16)$$

This formula could be programmed in merely a few code lines and can be deployed at the start of each new iterative solution process for solving a new system like Eq.(1.12).

3. IMPLEMENTATION DETAILS AND PERFORMANCE ANALYSIS

As F-COBRA-TF is programmed in Fortran90, the programming was done in Fortran90 and in OpenMP for the realization of the parallelized implementation. The main goal of the work has been to enable clear reductions of wall clock run times due to a realized capability to pursue F-COBRA-TF computations on parallel platforms (i.e. up to 12 parallel CPUs).
During a previous development project, a large part of the F-COBRA-TF code was parallelized already using OpenMP. The only remaining fully sequential component was the sparse linear solver which was available in two variants. The first one was based on direct Gaussian elimination, the second one was an iterative Krylov solver. The latter was indeed better suited for solving sparse linear systems, but was based on the SPARSKIT library [6], which turned out to be unsuitable for parallelization with OpenMP. Consequently, much priority was put on realizing, optimizing and parallelizing a new, in-house-developed implementation of the BiCGStab algorithm for solving the pressure equations in F-COBRA-TF.

Three different BiCGStab solution variants have been programmed and implemented; they differ only in how the preconditioning and the associated parallelized loops had to be arranged:

1. *BiCGStab classic* (no LDP) that solves the untransformed equation (1.1)

2. *BiCGStab RB-LDP* that solves the equation (1.2) featuring LDP preconditioning on a red-black grid

3. *BiCGStab ω-RB-LDP* that solves the ω-transformed and RB-LDP-preconditioned equation (1.12) on a red-black grid while using formula (1.16) for computing the optimum value for ω

3.1. Parallelization aspects for the three developed BiCGStab solution variants

Regardless of the applied preconditioning approach, it should never be a problem to parallelize the BiCGStab algorithm. In the pursued implementations for F-COBRA-TF, it was not difficult to add the needed OpenMP directives to the relevant matrix-vector multiplication loops. Nonetheless, any LDP setup inevitably introduces loop carried dependencies that may make loop parallelization nontrivial. Due to this, LDP is in fact applicable in parallelized mode only when certain system properties are fulfilled, such as when the computational grid can be partitioned according to a red-black (RB) structure. This always implies having to decompose matrix-vector multiplication loops into smaller subloops.

It should be pointed out that it is not always possible to impose RB grid partitioning in all F-COBRA-TF applications. In case this is not possible, the solution variant *BiCGStab classic* is to be used. A great advantage from a parallelization point of view is that BiCGStab classic is the easiest one to parallelize since no special loop structure connected with any dedicated LDP-measure is necessary. However, as soon as a basic LDP measure is possible, the LDP-variant is always the more attractive one from a mathematical, numerical and mostly also performance point of view. The latter is usually a general truth, in spite of a possibly somewhat higher implementation effort and a subsequent systematic increased OpenMP overhead in parallel execution mode. As pointed out already, this increased overhead is primarily connected with the necessity of working with decomposed loops (i.e. separated red and black loops) for handling the matrix-vector multiplications. For a number of important cases (such as the ATRIUM^{TM} 10 model), the RB partitioning was straightforward, so there the computational sequences could profit optimally from the realized preconditioning measures.

From the point of view of the computational cost of a single BiCGStab step for the different variants, the following is summarized:

- Each BiCGStab step comprises two matrix-vector multiplications (i.e. *matvecs*) consistent with the linear system form that is solved [4].

- In sequential computation mode, the RB-LDP matvec $(1\text{-}A_{LD})^{-1}\ A_{UD}\ x$ does *not* present more computational effort than the classic matvec $(A_{LD} + A_{UD})\ x$; however in parallel mode the RB-LDP structure for the matvec necessitates decomposed (i.e. separated red and black) loops; as the number of used parallel CPUs increases, this leads to a higher OpenMP overhead ; the latter effect may however become less significant in case of solving very large systems, as then the raw number crunching effort, even if distributed over many CPUs, can be expected to outweigh the computational overhead of inter-CPU data communications in OpenMP.

- The ω-RB-LDP matvec is more than twice as expensive as a classic matvec: in parallel mode the same kind of increased OpenMP overhead can be expected as for RB-LDP that may, to a certain degree, degrade (case-dependently) a computational advantage gained from a condition number enabled by ω-RB-LDP compared to RB-LDP.

- All developed BiCGStab solver variants have shown a satisfactory parallel scalability (cf. subsection 3.3).

As will be discussed in more detail in the upcoming section 3.4, the comparatively lowest run times (in both sequential and parallel execution mode) for the ATRIUM^{TM} 10 test case were achieved with the solver variants RB-LDP and ω-RB-LDP. Due to its comparatively lean matvec kernel, it was in fact the RB-LDP variant that was able to provide the very lowest overall run times (especially for the sequential run) for the ATRIUM^{TM} 10 test case.

3.2. General code optimization

Special purpose routines that have been written by following a standard recipe may give correct computational results from the start, but may also still need further adjustments before being optimally efficient from a run time point of view. With run time minimization being a continuous objective in code development, a performance-oriented code review was therefore pursued.

Within this framework, the first task had been to check and adapt array dimensions such that sequential rather than strided memory access is achieved. Since F-COBRA-TF is programmed in Fortran90, this meant that specific performance-critical multi-level loops (i.e. the ones that cover a substantial part of the total computational effort) should be structured in such a way that the innermost loop consists of a sweep over the first array index. Also the inverted approach is sometimes possible, i.e. modification of an array structure for better matching a given loop structure in the code. In any case, care was taken to adhere, as much as possible, to the known rule of having multi-loop structures coincide with the given array index sequences (i.e. the first index should correspond to the innermost loop, the last index should correspond to the outermost loop). In one specific case, this indeed also meant reshaping the index structure of a two-dimensional array for having it match the given loop structure in an optimal way.

The second pursued task was to merge numerous small loops into fewer larger loops since, due to improved processor cache reuse, this is usually beneficial to performance. In case of the BiCGStab algorithm, care was taken to realize a structure of the matrix-vector multiplication loops that adheres, as much as possible, to an optimized programming of the inner products.

3.3. Sequential and parallel run time measurements for the ATRIUM^{TM} 10 test case

The F-COBRA-TF executable was built with the Intel Fortran compiler (version 11.1.056). The run time reduction measurements were pursued on a dual processor 12-core computer server based on Intel Xeon 5670 processors.

The investigated ATRIUM^{TM} 10 test case comprised more than 7000 successive sparse linear system solution processes for a 3D sub-channel grid comprising about 10000 fluid mesh cells. Whereas for the SPARSKIT implementation, the fractional workload associated with linear system solutions was about 30 %, this fractional workload was reduced to about 10-15 % for the

in-house developed BiCGStab solvers. The following tables list the total run time reduction factors for the different computation scenarios and sizes of the parallel clusters.

Table I. Relative speedup factors (sf_1 / sf_2) for case ATRIUM™ 10, compared to the sequential run time with use of SPARSKIT (sf_1) and compared, per solution variant, to the individual sequential runs (sf_2)

CPUs	SPARSKIT	BiCGStab Classic	BiCGStab RB-LDP	BiCGStab ω-RB-LDP
1 CPU	1.00 / 1.00	0.88 / 1.00	1.12 / 1.00	1.04 / 1.00
6 CPUs	2.67 / 2.67	4.22 / 4.78	4.82 / 4.31	4.72 / 4.56
12 CPUs	3.20 / 3.20	7.39 / 8.37	7.73 / 6.90	7.87 / 7.60

Table II. Accumulated basic operations for case ATRIUM™ 10

	BiCGStab Classic	BiCGStab RB-LDP	BiCGStab ω-RB-LDP
Number of inner iterations	1.666.127	845.567	489.943
Number of matrix vector multiplications	3.332.254	1.698.443	1.989.008

3.4. Analysis of the observed run time differences

The observed wall clock time dependence on (i) the solution variant, and (ii) the number of used parallel CPUs, as observed for the ATRIUM™ 10 test case, is depicted in Figure 10, where it has been normalized against the sequential run time with use of SPARSKIT. The observed dependence is consistent with the properties pointed out already in section 3.1:

- Using the SPARSKIT solver, one gets a rather early saturation of the speedup (factor 2.5 up to 3) that can be achieved using parallelization. This was due to the fact that this solver was a remaining sequential part of an otherwise parallelized code (i.e. early Amdahl saturation since a significant chunk of computational work remained sequential).

- All of the developed BiCGStab solver variants show a clearly more satisfactory parallel scalability, i.e. a speedup factor of about 4.5 with 6 CPUs and of about 7.5 with 12 CPUs.

- The RB-LDP and ω-RB-LDP variants show a clearly steeper convergence error decay rate than the classical variant (without LDP) in terms of required numbers of BiCGStab steps (cf. Figure 11). Due to this, the number of necessary matrix-vector multiplications is substantially lower and this is obviously a run time reducing effect. The impact of this improvement is most obvious in sequential computation mode where, compared to the

classical variant, the RB-LDP and ω-RB-LDP variants require clearly less CPU-time for the linear system solutions.

- The variant ω-RB-LDP requires, by far, the lowest number of BiCGStab steps for achieving a fixed convergence accuracy target. However, for the final wall clock run times of the ATRIUM^{TM} 10 test case, this advantage (compared to RB-LDP) is neutralized by the clearly higher computational cost of the ω-RB-LDP matvec. *Due to its comparatively lean matvec kernel, it was in fact the RB-LDP variant that was able to provide the lowest overall run times for the ATRIUM^{TM} 10 test case.*

- Compared to the classical variant, the parallel scalability of both RB-LDP and ω-RB-LDP is clearly less favorable. This is due to the necessity of decomposing the matvec loops into separated red and black loops. This decomposition introduces additional implicit OpenMP barriers at the end of each partial loop, and the overall parallel overhead is increased by this. As pointed out already in section 3.1, this aspect can be expected to have higher significance for smaller problem sizes and lower significance for larger problem sizes.

Nonetheless, a clear overall computational advantage can be concluded for the developed BiCGStab variants RB-LDP and ω-RB-LDP. This conclusion emerges in spite of an also clearly observable phenomenon of an arising computational penalty. This penalty is connected with an increased OpenMP overhead for both RB-LDP and ω-RB-LDP compared to the classic approach.

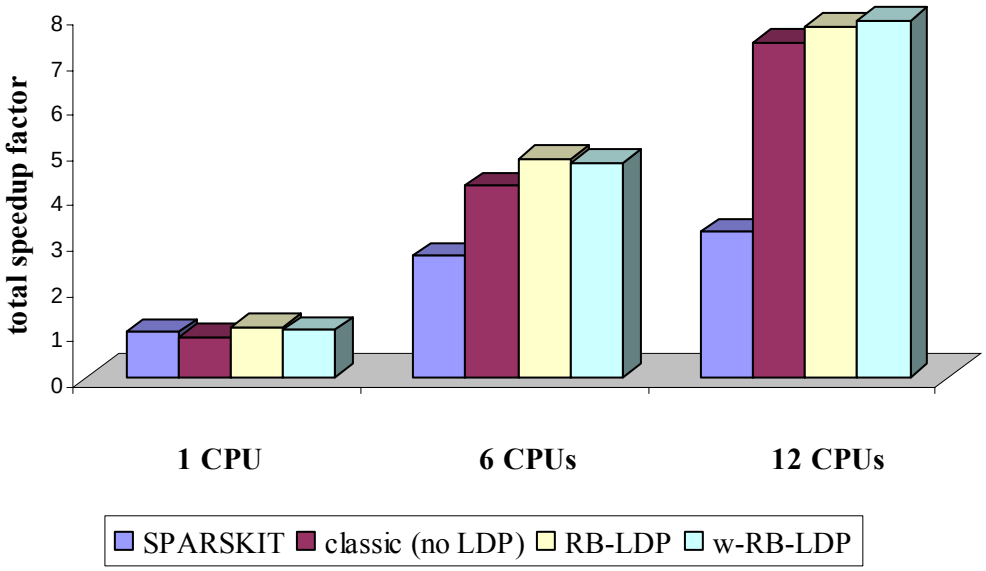

Figure 10. Comparison of total speedup factors (sf_1 as in Table I) for ATRIUM^{TM} 10 computation

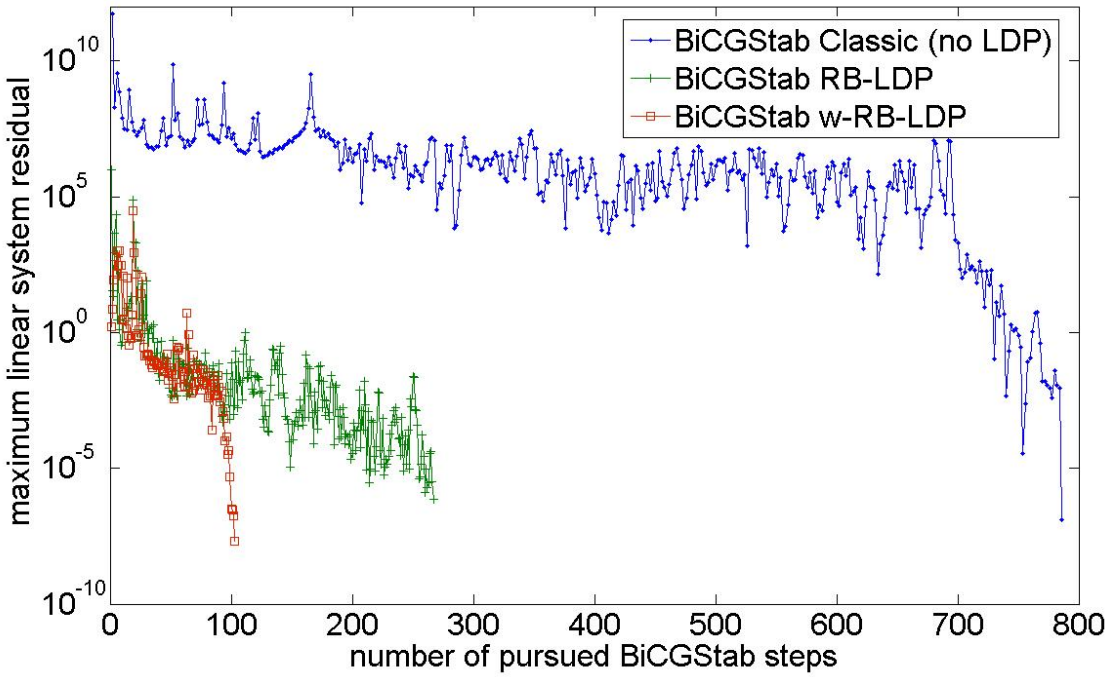

Figure 11. Comparison between the available new BiCGStab solvers for a representative sequence point in terms of basic convergence error decay properties (in terms of solution error evolution vs. number of pursued BiCGStab steps)

4. CONCLUSIONS

The in-house development of a preconditioned and parallelized BiCGStab solver has been pursued successfully in AREVA's advanced sub-channel code F-COBRA-TF. This solver can be run either in a sequential computation mode on a single CPU, or in a parallel computation mode on multiple parallel CPUs. Thus, F-COBRA-TF enables the computation of several thousands of successive sparse linear pressure system solutions with acceptable wall clock run times.

A number of computational verifications have confirmed that it may pay off generously to invest some development effort in the parallelized implementation of an appropriate preconditioning approach for the specific type of linear system to be solved. The latter serves to strengthen the diagonal dominance in what usually ends up being a slightly transformed linear system that nonetheless has the same, unique solution as the untransformed system. Due to this, the BiCGStab algorithm can be expected to convergence significantly faster in terms of required numbers of BiCGStab steps to be pursued for achieving a fixed convergence accuracy criterium. However, in case the preconditioning measure also implies a higher computational cost of a single BiCGStab step, this does not necessarily mean that the overall run time is always lower.

For F-COBRA-TF, different BiCGStab solution options have been developed. These differ from each other in terms of increasing diagonal dominances. However, these may be accompanied also by increasing computational costs associated with a single BiCGStab step. For parallel computations, an additional aspect on top of this is the phenomenon of increased OpenMP overhead. This is due to the decomposed loop structures that are typically necessary when wanting to benefit from a preconditioning property while also wanting to arrange the workload over multiple CPUs. Due to this, it is also possible that the solution variant which is the fastest in sequential computation mode may not be the fastest when run on a parallel platform, or not anymore be quite so much the fastest.

Nonetheless, all of the developed BiCGStab solver variants showed a satisfactory computational performance as well as a good parallel scalability. The comparatively lowest run times (in both sequential and parallel execution mode) for the investigated test case were, nonetheless, achieved clearly with the preconditioned solver variants. Out of these, the preconditioned variant featuring the leanest matrix-vector multiplication kernel proved to be the one to enable the very lowest wall clock run times.

REFERENCES

1. M. Glück, “Sub-channel analysis with F-COBRA-TF – code validation and approaches to CHF prediction.” *Nuclear Engineering and Design,* **237(6)**, pp. 655-667 (2007).
2. M. Glück, “Validation of the sub-channel code F-COBRA-TF Part I. Recalculation of single-phase and two-phase pressure loss measurements.” *Nuclear Engineering and Design,* **238(9)**, pp. 2308-2316 (2008).
3. M. Glück, “Validation of the sub-channel code F-COBRA-TF Part II. Recalculation of void measurements.” *Nuclear Engineering and Design,* **238(9)**, pp. 2317-2327 (2008).
4. H. A. Van der Vorst, “BiCGStab: a Fast and Smoothly Converging Variant of Bi-CG for the Solution of Non-symmetric Linear Systems”, J.Sci.Stat.Comput. Vol13, No.2, pp.631-644 (1992).
5. E.L. Wachspress, “Iterative Solution of Elliptic Systems”, Prentice Hall (1966).
6. Y. Saad, “SPARSKIT: A basic tool-kit for sparse matrix computations”, http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html