New Generation Iterative Solvers in LS-DYNA®

Cleve Ashcraft, Roger Grimes, Robert Lucas, François-Henry Rouet
Livermore Software Technology

Abstract

Iterative solvers for sparse linear systems are used as the default option in a few places in LS-DYNA, e.g., thermal analysis and incompressible fluid flow. They are also available as a non-default option for implicit mechanics, electromagnetics, and acoustics. Until now, our suite of iterative solvers was limited to a few simple solvers (Conjugate Gradients and GMRES), and a few simple preconditioners. We recall that a preconditioner is an approximate inverse of the matrix (e.g., the stiffness matrix) that aims at improving convergence. Our preconditioners were limited to simple techniques like diagonal scaling and basic domain decomposition techniques that discard the coupling terms between processors, in MPP. Problems from customers are growing faster than memory size, making it difficult to use direct solvers. They are also often too numerically challenging to use simple iterative solvers, in particular in implicit mechanics. This has pushed us to revisit our suite of iterative solvers and preconditioners. In particular, we have been investigating the use of Block Low-Rank factorizations (BLR) and the use of Algebraic Multigrid (AMG). In the talk, we will compare these new options across all the different applications that make use of linear solvers. We will discuss convergence, memory usage, and scalability. For end users, the takeaway will be a better understanding of which solver options to use for different kinds of problems, and what to expect from them.

Introduction

Linear solvers are ubiquitous in LS-DYNA and simulation codes in general. They are broadly classified into two groups of algorithms. *Direct solvers* are essentially "exact" solvers, typically based on Gaussian Elimination; they factor the input matrix A into triangular factors (A = L U factorization in the non-symmetric case, A = L D L^T factorization in the symmetric case) and compute the solution by triangular solves. They are robust and accurate but their cost (time and memory) can be a hurdle. On the other hand, *iterative solvers* try to converge to the solution of the problem by generating a sequence of approximate solutions. They are generally cost-effective, since the main computational kernel is a sparse matrix-vector product, but they are less robust than direct solvers and can fail to converge for difficult problems. The Conjugate Gradients algorithm of Hestenes and Stiefel [1] and the Generalized Minimal Residual (GMRES) method of Saad and Schultz [2] are the most well-known iterative methods. One way to make iterative solvers more robust is the use of *preconditioners*; a preconditioner P is a form of approximation of A such that P^{-1} A x = P^{-1} b is easier to solve than the original problem A x = b. The crux of the matter is to find a matrix P that helps the iterative solver to converge but such that solving with P (applying P^{-1}) is fast. Finding good preconditioners is often very problem dependent and it is admittedly as much of an art as it is a science.

In LS-DYNA, we have historically focused on direct solvers. This is because, for many years, the main driver was implicit mechanics, which is an area where iterative solvers tend to fail, especially for applications like metal forming and metal stamping. For a long time, only the thermal solver made use of iterative solvers as the default approach; this is because our thermal problems are numerically well-behaved most of the time, especially in transient analysis. However, new applications and areas of research have pushed us to revisit iterative solvers and preconditioners:

- The incompressible fluid flow module of LS-DYNA solves the Incompressible Navier-Stokes equations using a predictor-corrector approach in which the momentum (or velocity) equations and the pressure equations are solved separately. The momentum equation is non-symmetric and is solved with GMRES. The pressure equation is symmetric and is solved with Conjugate Gradients. The latter equation is numerically difficult and has prompted us to explore many different preconditioners.

- The electromagnetism module of LS-DYNA solves the eddy current approximation of the Maxwell equations with finite elements. The resulting system is solved using a direct solver by default, but problems coming from customers have grown fast in recent years, for example simulations of car batteries and problems in electrophysiology like heart simulations. Having efficient iterative solvers for these problems is therefore more and more important. Note that the electromagnetism solver also use boundary elements to solve the surface of the conductors, instead of meshing the surrounding air. Finite element formulations lead to "sparse" linear systems, meaning that the corresponding matrix has few nonzero elements. This is due to the locality of interactions across the mesh; nodes only interact to their neighbors, not with every node in the physical domain. On the other hand, boundary element formulations lead to "dense" (or "fully populated") systems where the matrix is not sparse. Here, we only focus on sparse problems from finite elements. Our dense linear solvers for boundary elements are described in detail in another publication [3].
- New applications from multi-scale mechanics, in particular problems involving Representative Volume Elements (RVEs) have been pushing direct solvers to their limit, in terms of cost. We describe such an application in detail in one of the following sections.
- Isogeometric analysis (IGA) is another area where iterative solvers might play a role. Matrices arising from IGA tend to be less sparse than matrices coming from standard finite elements, which could make it harder to use direct solvers.

In the next section, we remind the reader what iterative solver options are currently available in LS-DYNA. The following section presents recent developments.

Current iterative solvers in LS-DYNA

Iterative solvers

Most of the matrices that we deal with are symmetric and indefinite; their null space (zero eigenmodes) correspond to rigid body modes in implicit mechanics, or the to fact that the electric potential field is defined up to constant in electromagnetism, etc. However, when appropriate boundary conditions (constraints) are imposed (e.g., prescribed motion or contacts in implicit mechanics), we use the Constraint Method [4] (also referred to as the null space method, or direct elimination of constraints) to form a reduced linear system corresponding to the independent degrees of freedom (dofs). This removes the null space of the matrix, and we obtain a positive definite system that can be solved with the Conjugate Gradients algorithm. This is the default solver for thermal simulations, and for the pressure solve of the incompressible fluids code. It can also be used for implicit mechanics and electromagnetics.

A few applications generate nonsymmetric matrices: rotational dynamics with gyroscopic effects (ISTFNS=3 in *CONTROL_IMPLICIT_DYNAMICS), conjugate heat transfer problems with a monolithic coupling between the structure and the fluid (CTYPE=0 in *ICFD_CONTROL_CONJ), … For these applications, GMRES is our iterative solver of choice.

Preconditioners

As mentioned in the first section, preconditioners are a critical ingredient of iterative solvers. Our original set of preconditioners includes:

- Diagonal preconditioner (also known as *Jacobi preconditioner*, or *diagonal scaling)*: this is simply the diagonal of A; P=diag(A). This is the simplest of preconditioners. It is inexpensive; computing P^{-1} x for a given x costs n floating-point operations (with n the size of the problem) and doesn't require any inter-processor communications, since P is diagonal. Obviously, this is not a very powerful preconditioner, but it can be enough for simple problems. For example, this is the default option for thermal simulations (SOLVER=12 in *CONTROL_THERMAL_SOLVER). It is also available in implicit mechanics (LSOLVR=22 in *CONTROL_IMPLICIT_SOLVER).
- A block diagonal preconditioner, where the blocks are defined by the MPP decomposition of the problem. In MPP, every processor owns a subset of columns of the matrix. Therefore, every processor owns a diagonal block (corresponding to a local problem on the subdomain owned by the processor), and off-diagonal terms that define the coupling between different subdomains. Our block diagonal preconditioner simply performs a local solve on each diagonal block (one per processor), ignoring the coupling terms. To mitigate cost, these local solves are themselves approximate solves (therefore there are two levels of approximation here), and we have four different flavors, inspired by traditional preconditioning techniques:
  - Symmetric Gauss-Seidel (SGS): SOLVER=13 for thermal, LSOLVR=23 for implicit. Denote B the diagonal block owned by a given processor, and consider the splitting B = L + D + U, where L is the lower triangular part of B (excluding the diagonal), D is the diagonal of B, and U is the upper triangular part of B (excluding the diagonal), equal to L^T in the symmetric case. The approximate local solve is defined by $(D + U)^{-1} D (L + D)^{-1}$.
  - Symmetric Successive Over Relaxation (SSOR): SOLVER=14 for thermal, LSOLVR=24 for implicit. This is an extension of the Gauss-Seidel approach. For a given parameter w, the approximate local solve is defined by $(2 w – 1) (w D + L)^{-1} D (L + w D)^{-1}$. For w = 1, SSOR and SGS are identical.
  - Zero-fill incomplete factorization: SOLVER=15 or 16 for thermal, LSOLVR=25 or 26 for implicit. The local diagonal block B is sparse; if we were to compute an exact factorization in order to perform an exact local solve, the number of nonzeros in the factors of B would in general be much larger than the number of nonzeros in B itself, a phenomenon known as *fill-in.* A zero-fill incomplete factorization is an approximate factorization where fill-in terms are discarded. The two options (15 and 16 for thermal, 25 and 26 for implicit) correspond to two different implementations; the second option uses more storage but applying the preconditioner is faster.
  - Threshold-based incomplete factorization: SOLVER=18 for thermal, LSOLVR=27 for implicit (from R12.0). This is similar to the previous option, but fill-in entries are discarded based on their magnitude (entries above a certain numerical threshold are kept while those below the threshold are discarded).

  The main interest of this block diagonal approach is that it does not involve inter-processor communication, since each processor (MPI rank) only performs a local solve, independently of the other processors. However, this means that coupling terms between subdomains are ignored. This also means that the preconditioner becomes weaker when the number of processors increases, since there are then more diagonal blocks (of smaller size), and more coupling terms are ignored. In the limit, when the number of processors is "infinite" (as large as the number of unknowns), this block diagonal preconditioner becomes a simple diagonal preconditioner. Of the four variants, the threshold-based incomplete factorization (SOLVER=18, LSOLVR=27) is usually the most robust. This is the option we use for pressure solves in the incompressible fluid solver, with the dropping tolerance set to 10^{-3}.

Problems from customers have been growing faster than memory size, making it harder to use direct solvers, which are very memory consuming. We are now seeing problems with hundreds of millions of degrees of freedom in multiple areas of applications: automotive, gas turbines, and biomedical simulations. Furthermore, simple preconditioners like diagonal scaling or block diagonal preconditioners are often not powerful enough to handle complex physics, and block diagonal preconditioners quickly become ineffective when using hundreds or thousands of MPI ranks, which is more and more common.

New generation of iterative solvers and preconditioners

MINRES

As mentioned above, the majority of our problems are symmetric positive definite (after eliminating the constraints), and the Conjugate Gradients algorithm can be used. We also have nonsymmetric problems which can be solved with GMRES. But we also have a few symmetric indefinite problems like *inertia relief,* which allows the analysis of models that have unconstrained rigid body modes. Such problems could be solved using GMRES, ignoring symmetry, but GMRES can be costly since it needs to maintain an orthogonal *Krylov space*. An alternative is to use the Minimal Residual (MINRES) method of Paige and Saunders [5], which targets symmetric indefinite systems, and of which GMRES is a generalization. Contrary to GMRES, MINRES does not need to maintain a Krylov space. However, even though MINRES can solve indefinite problems, it requires a *positive definite preconditioner*, which can be a major hurdle. If the input matrix A is indefinite, then a good preconditioner, which typically is a good approximation of A, will likely be indefinite too, and MINRES will break down.

We implemented MINRES and tried several inertia relief problems from automotive applications. We found out that we could not achieve convergence when using weak preconditioners (such as some of the ones described above); on the other hand, strong preconditioners (such as the ones described in the next sections) caused MINRES to break down, because they were not positive definite. There are standard ways of modifying preconditioners to force positive definiteness, but we have not explored these in depth. For now, we choose not to add MINRES to the set of iterative solvers available to users.

Block Low-Rank factorizations (BLR)

In many applications, matrices have low-rank (singular) off-diagonal blocks. This is true for the dense matrices arising in boundary element methods, and this is also true in finite element methods; stiffness matrices are sparse, but factoring a sparse matrix can be cast a sequence of dense factorizations of smaller, intermediate matrices, and these matrices have low-rank blocks. Low-rank blocks are often referred to as *data sparse*, because they can be compressed using a singular value decomposition or a rank-revealing factorization. A low-rank m x n matrix B can be compressed into B = X Y, where X is m x r (“tall and skinny”) and Y is r x n (“short and wide”), with r the rank of B. If r is small enough, then storing X and Y is cheaper than storing B, and operating on X and Y is cheaper than operating on B. The compression can be exact and used to accelerate exact solvers, or it can be approximate (guided by a given threshold) and used to design approximate representations and preconditioners.

This observation gave rise to the field of Hierarchical Matrices [6]. We have been interested in a subclass of Hierarchical Matrices called Block Low-Rank matrices (BLR). This is a simple but effective representation where a matrix is partitioned following a regular partitioning of the rows and columns; off-diagonal blocks are compressed independently of each other. A BLR representation is illustrated below.

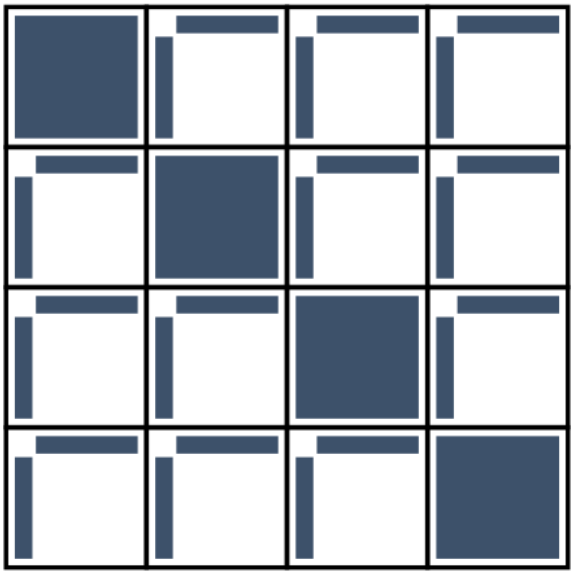

Figure 1: Block Low-Rank representation of a dense matrix; off-diagonal blocks are low-rank and represented as a product of two matrices to save storage and operation cost.

We have used BLR representations in our dense solvers for electromagnetics and acoustics since 2004. In 2010, we started collaborating with the MUMPS team. MUMPS [7, 8] is an open-source sparse direct solver that implements the multifrontal method of Duff and Reid [9]. MUMPS now has Block Low-Rank algorithms to speed-up the factorization and solution phase; by tuning the threshold used to compress blocks, it can be used a direct solver or a preconditioner.

The figure below illustrates the performance of MUMPS for two very different models: a Representative Volume Element (RVE) problem described in the next section, and a 105M degrees of freedom whole jet engine model provided by Rolls-Royce [10]. In each figure, the leftmost bar ("FR") is the performance of the standard solver, without BLR approximations ("full-rank" mode). Then, we switch to the BLR mode, which we use as a preconditioner for Conjugate Gradients. When the compression threshold is close to machine precision (left-hand side of each figure), we have an "almost direct" solver, which converges in a couple of iterations. As the threshold gets closer to 1 (right-hand side of each figure), we have a more and more aggressive preconditioner; performance improves, but sometimes convergence is lost. The value of the compression threshold which provides the best performance is problem dependent, but we found (across a large set of problems, not shown here) that 10^{-6} is often a good tradeoff. The figure also illustrates that for some problems we get significant gains from BLR (RVE case on the left-hand side), while sometimes we only get marginal gains (Jet Engine model on the right-hand side).

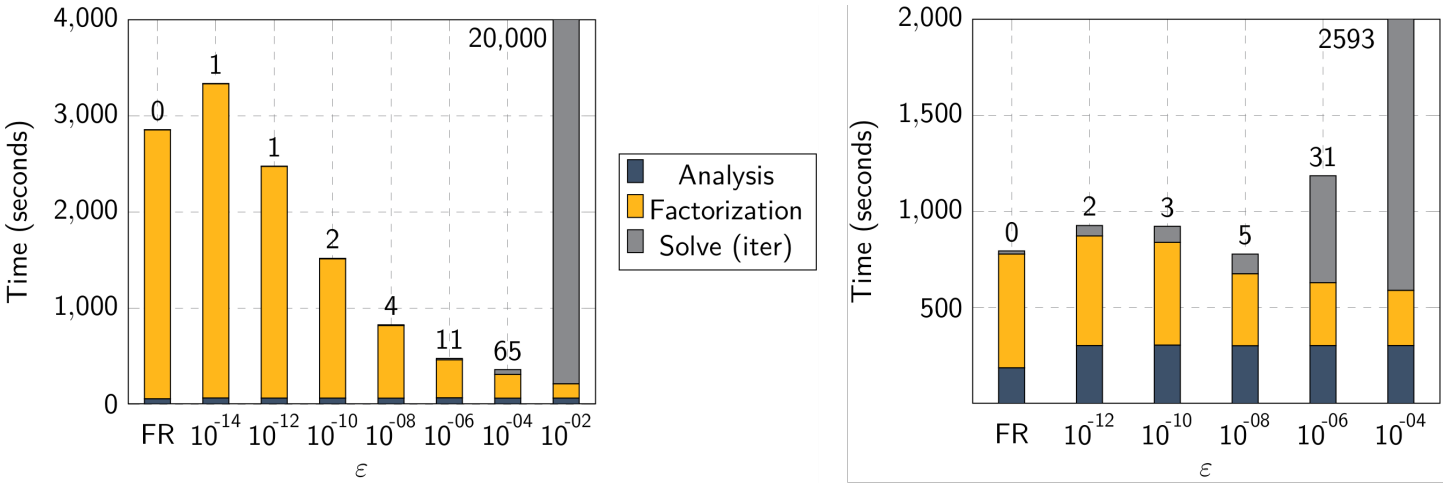

Figure 2: Performance of MUMPS-BLR (used as a preconditioner for Conjugate Gradients), as a function of the compression threshold, for two problems: 3M dof hyperelastic rubber RVE (left-hand side) and 105M dof jet engine model (right-hand side).

The next figure illustrates the scalability of the factorization and triangular solve when one increases the number of MPI ranks. For this problem (jet engine model), the BLR factorization and solve scale as well as the standard ("full-rank") factorization and solve do.

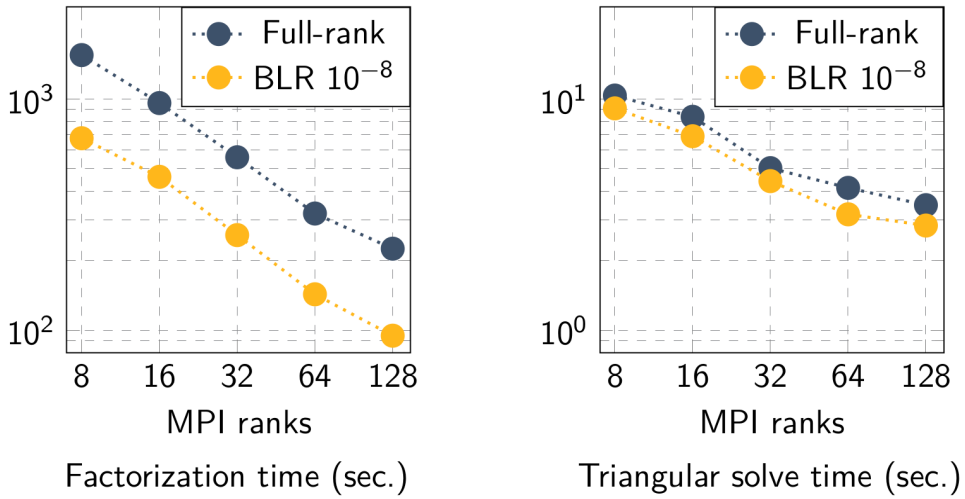

Figure 2: Scalability of MUMPS-BLR for a 105M dof jet engine model.

The last figure is a complexity experiment. There are theoretical results that describe the asymptotic complexity of sparse direct solvers and BLR solvers for regular 2D and 3D grids, but the vast majority of our problems are unstructured meshes. The figure uses three versions of the jet engine model prepared by Rolls-Royce: 11M nodes (33M degrees of freedom), 35M nodes (105M dofs), and 67M nodes (200M dofs). The figure shows that the number of floating-point operations for the full-rank factorization seems to increase slightly more than quadratically, while the increase is less than quadratic for the BLR factorization.

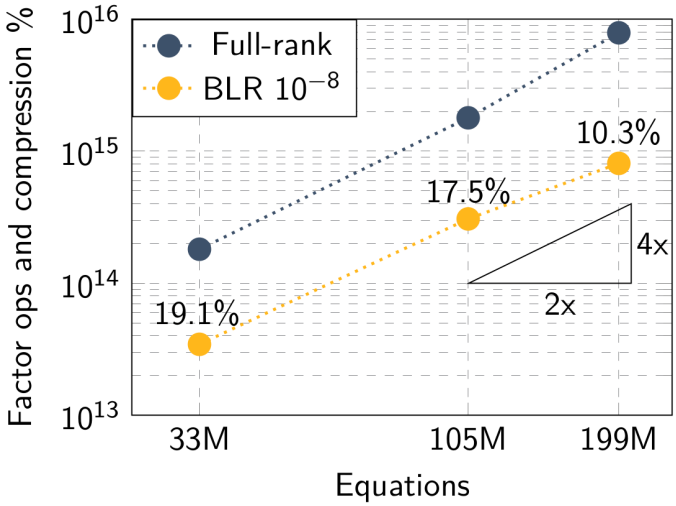

Figure 3: Complexity experiments for three versions of the same model, with MUMPS-BLR.

As of R11.1, MUMPS and its Block Low-Rank feature can be used for implicit mechanics problems (LSOLVR=30 in *CONTROL_IMPLICIT_SOLVER; the BLR compression threshold is specified using the DROPTOL parameter of the same keyword). It will be available as a thermal solver in R12.0. We are also using the Block Low-Rank mode of MUMPS in our implementation of the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) eigensolver [11] (EIGMTH=102 in *CONTROL_IMPLICIT_EIGENVALUE).

Block Low-Rank techniques are very promising. They are very robust (as long as the compression threshold is set to a reasonable value). Since they are accelerations of direct solvers, they sometimes suffer from similar problems (cost of the analysis phase, memory usage), but there has been tremendous progress in recent years. We are in the process of integrating low-rank techniques to our in-house multifrontal code MF2. We are also evaluating other classes of low-rank techniques and other low-rank solvers, such as the Hierarchically Semi-Separable (HSS) techniques implemented in STRUMPACK [12, 13].

Algebraic Multigrid (AMG)

Algebraic Multigrid (AMG) methods [14] construct a hierarchy of coarse problems to accelerate a standard iterative method, and they aim at obtaining linear complexity. They have been widely successful for many applications, in particular fluid dynamics, and they constitute a very large and active field of research. We provide a simplistic description here and refer the reader to the extensive literature about AMG. The main steps of an AMG method are:

1. Smoothing (relaxation): apply a few iterations of a simple iterative method like Gauss-Seidel or SSOR to reduce high-frequency errors from an initial guess.
2. Restriction (downsampling): build a coarse problem and transfer the residual error on that coarse problem.
3. Solve the coarse problem.
4. Prolongation (interpolation): transfer the coarse solution back to the fine problem.

This sequence defines a simple 2-level method, but step 3 can be replaced by a recursive application of steps 1-4, defining a multilevel hierarchy of coarse problems, until the coarse problem is small enough to be solved efficiently.

Our motivation for using AMG came from an application in material design, in collaboration with the Yokohama Rubber Company. The model is a Representative Volume Element (RVE) (sometimes also referred to as a *unit cell*) that consists of a cube of hyperelastic rubber with very stiff inclusions; periodic boundary conditions are used to simulate a larger macrostructure [15]. The discretization is a cubic grid of solid hexes. This kind of configuration is a worst-case scenario for direct solvers, because their memory usage grows as $O(n^{4/3})$ and their floating-point operation count grows as $O(n^2)$ for regular 3D grids, with n the number of degrees of freedom. The increase is usually less dramatic for 2D models or "2.5D" models that are relatively hollow, like car bodies and jet engines. With our in-house direct solver, and well as with the external direct solver MUMPS, we were not able to solve further than a 150x150x150 grid (3.4M nodes, 10.1M degrees of freedom), but the goal for this application is to be able to solve at least 300x300x300 (81M degrees of freedom).

We experimented with the public library Hypre [16]. Hypre is a library of iterative solvers and preconditioners, and it contains an implementation of Algebraic Multigrid, BoomerAMG [17]. We used BoomerAMG as a preconditioner for Conjugate Gradients. The table compares performance of our in-house direct solver ("MF2") against MUMPS (in Block Low-Rank mode, as described in the previous section), and BoomerAMG, for a 150x150x150 problem. For this model, AMG outperforms the direct solver and the BLR approach. All the simple preconditioners mentioned in the previous section failed for this problem.

Solver	Iterations	Time (sec)
MF2	-	6,718
MUMPS-BLR	18	2,692
Hypre-BoomerAMG	716	503

Figure 4: Performance of different solvers for a 150x150x150 RVE.

The next figure illustrates the linear complexity of the AMG approach. Here the grid size ranges from 50x50x50 to 250x250x250, and run time grows linearly with problem size.

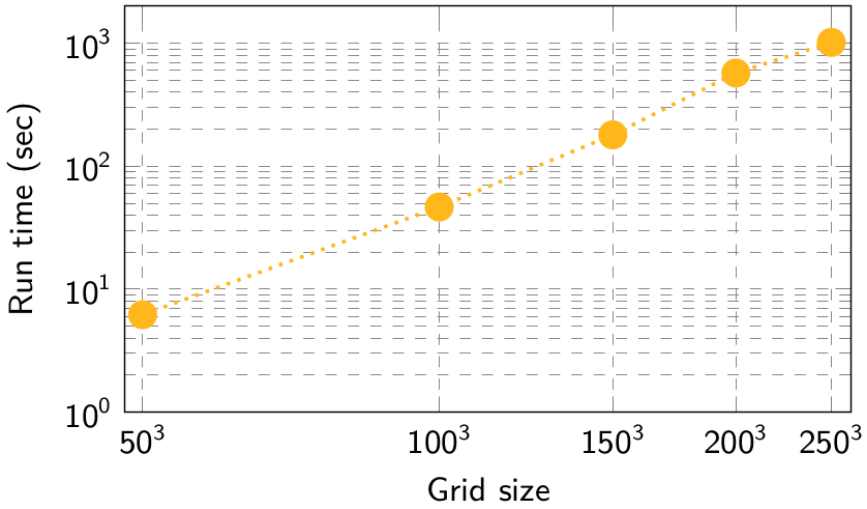

Figure 6: Performance of BoomerAMG for different RVE grids.

It is important to note that it took a long time to get these results. AMG codes have many options and finding the best set of parameters for a given problem can take a lot of trial and error. Furthermore, even though AMG methods can be used in a purely algebraic setting (i.e., the only input are a matrix and a right-hand side vector), they can greatly benefit from additional information, such as the mapping from equations to dof type, which requires an additional integration effort.

We plan on experimenting with BoomerAMG (and AMG methods in general) further, with different classes of problems, in particular the pressure equation of the incompressible fluid solver.

Conclusion and future work

Even though iterative solvers have robustness issues, they are an important tool of our suite of solvers, both for problems that are known to converge easily (e.g., transient thermal transfer), and for very large scale problems that are intractable to direct solvers (e.g., very large Representative Volume Element problems). We have explored the use of the well-known Algebraic Multigrid algorithm (through the open source package BoomerAMG which is part of the Hypre library), and we are actively exploring the area of low-rank factorizations, in particular Block Low-Rank techniques. AMG turns out to be useful for very specific applications and we are continuing to evaluate its applicability to a wider range of problems. Low-rank methods have already demonstrated their usefulness and their robustness, and work is in progress to reduce their computational requirements further. As of LS-DYNA R11.1, BLR techniques are available (through the MUMPS package) for implicit mechanics (LSOLVR=30 in *CONTROL_IMPLICIT_SOLVER). As of LS-DYNA R12.0, they will be available for challenging thermal problems (SOLVER=19 in *CONTROL_THERMAL_SOLVER). In a future version of LS-DYNA, they will be made available to users as documented solver options for Incompressible CFD and Electromagnetism.

We continue to explore different ideas for iterative solvers and preconditioners, and we are also exchanging ideas and software packages with the different Ansys groups.

Acknowledgments

We thank Victor Magri (Lawrence Livermore National Laboratory) for his help with Hypre and BoomerAMG. We thank Patrick Amestoy and Jean-Yves L'Excellent (MUMPS Technologies) for their help with MUMPS.

References

[1] M. R. Hestenes and E. Stiefel. "Methods of conjugate gradients for solving linear systems." *Journal of research of the National Bureau of Standards* 49, no. 6 (1952): 409-436.
[2] Y. Saad and M. H. Schultz. "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems." *SIAM journal on scientific and statistical computing* 7, no. 3 (1986): 856-869.
[3] P. L'Eplattenier, G. Cook, C. Ashcraft, M. Burger, J. Imbert, and M. Worswick. "Introduction of an electromagnetism module in LS-DYNA for coupled mechanical-thermal-electromagnetic Simulations." *Steel research international* 80, no. 5 (2009): 351-358.
[4] C.-T. Tsai and B. A. Szabo, "The constraint method: a new finite element technique." NASA Technical Memorandum, NASA, TM X-2893 (1973): 551-568.
[5] C. C. Paige and M. A. Saunders. "Solution of sparse indefinite systems of linear equations." *SIAM journal on numerical analysis (SIMAX)* 12, no. 4 (1975): 617-629.
[6] W. Hackbusch. "A sparse matrix arithmetic based on H-matrices. Part I: Introduction to H-matrices." *Computing* 62, no. 2 (1999): 89-108.
[7] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. "A fully asynchronous multifrontal solver using distributed dynamic scheduling." SIAM Journal on Matrix Analysis and Applications 23, no. 1 (2001): 15-41.

[8] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. "Performance and scalability of the block low-rank multifrontal factorization on multicore architectures." *ACM Transactions on Mathematical Software (TOMS)* 45, no. 1 (2019): 1-26.
[9] I. S. Duff and J. K. Reid. "The multifrontal solution of indefinite sparse symmetric linear." *ACM Transactions on Mathematical Software (TOMS)* 9, no. 3 (1983): 302-325.
[10] T. A. Simons, J. S. Ong, R. F. Lucas, F.-H. Rouet, R. Grimes, E. Gulyeruz, S. Koric, T.-T. Zhu, and J. Dawson. "Reducing the Time-to-Solution for Finite Element Analysis of Gas Turbine Engines." In *AIAA Propulsion and Energy 2019 Forum*, p. 44-56. 2019.
[11] A. V. Knyazev. "Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method." SIAM journal on scientific computing 23, no. 2 (2001): 517-541.
[12] F.-H. Rouet, X. S. Li, P. Ghysels, and A. Napov. "A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization." *ACM Transactions on Mathematical Software (TOMS)* 42, no. 4 (2016): 1-35.
[13] P. Ghysels, X. S. Li, F.-H. Rouet, S. Williams, and A. Napov. "An efficient multicore implementation of a novel HSS-structured multifrontal solver using randomized sampling." *SIAM Journal on Scientific Computing* 38, no. 5 (2016): S358-S384.
[14] A. Brandt, S. McCormick, and J. Huge. "Algebraic Multigrid (AMG) for sparse matrix equations." *Sparsity and its Applications* 257 (1985).
[15] C. T. Wu and M. Koishi. "Three-dimensional meshfree-enriched finite element formulation for micromechanical hyperelastic modeling of particulate rubber composites." *International journal for numerical methods in engineering* 91, no. 11 (2012): 1137-1157.
[16] R. D. Falgout, U. Meier Yang. "Hypre: A library of high performance preconditioners." In *International Conference on Computational Science*, pp. 632-641. Springer, Berlin, Heidelberg, 2002.
[17] U. Meier Yang. "BoomerAMG: a parallel algebraic multigrid solver and preconditioner." *Applied Numerical Mathematics* 41, no. 1 (2002): 155-177.