Linear Equation Solvers
Gerd-Jan Schreppers, January 2011

Abstract: This paper explains the procedures for solving linear set of equations in DIANA. In linear and implicit nonlinear finite element analysis the required memory and analysis time for large models is strongly depending on the chosen procedure for solving the set of linear equations. DIANA offers both direct and iterative equation solvers. It is explained which procedures should be applied for different types of models. In nonlinear analysis specific techniques such as sub-structuring and iterative corrections are explained. The effect on computation time of parallel processing is illustrated by several examples.

Introduction
DIANA offers the use of four different equation solvers:
- Automatic solver
- PARDISO solver
- Out-of-core direct solver
- Iterative solver

In nonlinear analysis some of these solvers can be used in combination with sub-structuring techniques.

The PARDISO solver and ITERATIVE solver offer parallel processing options.

Automatic Solver
By default, DIANA will automatically choose an optimal solution procedure which is initially based on a Sparse Cholesky method. If a factorization from a previous Sparse Cholesky solution exists, the automatic procedure uses this as a preconditioner in an iterative process. This automatic procedure does not require any additional SOLVE commands. However, for special cases you may customize the solution procedure via AUTOMA commands, such as for sub-structuring.

A general approach to solve the linear system of equations is first to factorize \( K \):

\[
\begin{align*}
K & \rightarrow LU \\
K & \rightarrow LDU
\end{align*}
\]

where \( L \), \( U \) and \( D \) are lower, upper and diagonal matrices respectively. This process is known as LDU decomposition. After the factorization, the solution vector \( u \) can be computed by successive forward and backward substitution to solve the triangular system of equations

\[
\begin{align*}
L w &= f \\
DUu &= w
\end{align*}
\]

Under the assumption that \( K \) is a positive definite symmetric matrix the decomposition becomes \( LL^T \) or \( LDL^T \).

PARDISO Solver
For Intel based Windows and Linux platforms DIANA offers the Intel PARDISO, parallel direct sparse solver. Intel Math Kernel Library (Intel MKL) provides a direct sparse solver PARDISO which can be used for solving real symmetric and structurally symmetric sparse linear systems of equations.

The PARDISO solver shows both a high performance and memory efficient usage for solving large sparse symmetric and un-symmetric linear systems of equations by shared multiprocessors. The solver uses a combination of left- and right-looking super-node techniques. For sufficiently large problems, the scalability of the parallel algorithm is nearly independent of the shared-memory multiprocessing architecture. If the user has not explicitly set the number of threads then this value can be set by the operating system to the available numbers of cores on the system. It is therefore always recommended to control the parallel execution of the solver by explicitly specifying the maximum number of threads that shall be used. If less cores are available then specified, the execution may slow down instead of speeding up.

Out-of-core Direct Solver
DIANA offers an out-of-core direct solution method based on Gauss decomposition. This solver may be useful for situations in which the matrix data cannot fully be taken in background storage. In such situations other solvers may fail to produce a solution due to a lack of memory. The swapping of back-ground storage by the out-of-core solver leads to longer analysis times in comparison with other solvers.
Iterative Solver

The direct solution method has some drawbacks. The most important one is that the background storage requirements can be extremely high for large three-dimensional problems. Another disadvantage is that the Gauss decomposition without pivoting (i.e., interchanging rows and columns) is not numerically stable if the stiffness matrix is not positive definite. For these reasons two iterative methods are available in DIANA as alternatives for the direct solution method.

The common idea of all iterative methods for solving the linear system of equations $Ku = f$ is to generate a sequence of approximations $u_i$ to the solution vector $u$ via the recursion

$$u_{i+1} = u_i + y_i Q(f - Ku_i)$$

in which $Q$ is the preconditioning matrix or the preconditioner. In some way, $Q$ should resemble the inverse of the stiffness matrix $K$. Two algorithms to compute the iteration parameters have been implemented in DIANA: the Conjugate Gradient method (CG) and the Generalized Minimal Residual algorithm (GMRES). DIANA uses CG for symmetric matrices and GMRES for nonsymmetric matrices.

The Conjugate Gradient method, is currently the most popular and probably the best iterative method for systems with a symmetric positive definite stiffness matrix, for example all linear elastic problems. The CG algorithm generates $y_i$ such that all residuals $r_i$ are perpendicular. By making clever use of the symmetry of $K$ it is possible to orthogonalize the residual $r_i$ against all previous residuals by making use of only the residuals of the two previous iterations. If the stiffness matrix is non-symmetric or indefinite, i.e., has negative eigenvalues, the Conjugate Gradient method need not converge.

The GMRES method converges even if the stiffness matrix is not positive definite, for instance if the stiffness matrix is nonsymmetrical. The iteration parameters are computed by orthogonalizing the residual explicitly against all previous residuals. To be able to do this, all residuals must be stored. Moreover, the number of computations per iteration increases since the orthogonalization process becomes more expensive every iteration. Therefore the iteration is restarted after a fixed number of residuals have been added to the basis.

Incomplete LU-decomposition

By default DIANA applies Incomplete LU-decomposition preconditioning, generally known as ILU preconditioning. The idea of ILU preconditioning is to approximate the system matrix $K$ by the product of a lower diagonal matrix $L$ and an upper matrix $U$

$$Q^{-1} = L \cdot U = K$$

If the factorization is carried out exactly, we get a direct solution method. The disadvantage of the exact factorization is that fill-in occurs: the matrices $L$ and $U$ contain far more non-zero entries than the original matrix $K$. In the ILU approximation we try to restrict the fill-in of $L$ and $U$. First we limit the fill to the sparsity pattern of $K$, i.e., $L_i \neq 0$ and $U_i \neq 0$ only if $K_i \neq 0$. The ILU decomposition is uniquely defined by

$$Q_{ij}^{-1} = \sum L_{ik} \cdot U_{kj} \quad \text{if} \quad K_{ij} \neq 0$$

If the setup of $Q$ fails, or if the subsequent iteration does not converge, we improve the pre-conditioner by allowing more fill-ins. Therefore we use a drop-tolerance strategy: non-zero elements are only included in the incomplete factors if they are larger than a given threshold parameter. This threshold parameter is determined adaptively: we decrease it until the iteration has converged. We notice that we obtain the exact factorization if the drop tolerance is small enough.

Jacobi Preconditioning

The most simple and probably the most widely used preconditioning technique is to scale the stiffness matrix with a diagonal matrix $D$. For problems with a diagonally dominant stiffness matrix the choice $D$ equal to $\text{diag}(K)$, and hence $Q$ equal to the inverse of $\text{diag}(K)$, is both natural and good. This preconditioner is known as Jacobi preconditioning or diagonal scaling.

The natural choice for the termination criterion of the iterative process is based on the reduction of the residual $r_i$,

$$r_i = f - Ku_i$$

with $u_i$ the approximation of the solution $u$ after $i$ iterations. The iteration is stopped if

$$r_i < \epsilon f$$

with $\epsilon$ a user-defined tolerance.

The iterative solvers can be used in combination with sub-structuring.
Substructuring

By default DIANA investigates whether substructuring in a structural nonlinear analysis with Module NONLIN could be profitable. If so, then sub-structuring will be applied automatically. In other types of analysis, for instance linear structural or potential flow, no sub-structuring is applied by default.

Substructuring is a standard technique in Finite Element Analysis. The basic idea is to treat a group of elements as a single substructure (super-element). The use of substructures is attractive in various cases. For example, if many elements in a nonlinear model behave linearly, these elements can be put in a substructure. The internal degrees of freedom in the substructure are then removed by static condensation.

Suppose that we have two substructures that only contain linear elements. The degrees of freedom in the substructures can then be divided in internal degrees of freedom and interface degrees of freedom. After a proper reordering, the stiffness matrix of this system can be written as

\[
\begin{bmatrix}
A_1 & 0 & B_1 \\
0 & A_2 & B_2 \\
B_1^T & B_2^T & C
\end{bmatrix}
\]

where \(A_1\) and \(A_2\) are the submatrices representing the connectivity of the internal degrees of freedom of the substructure. The rectangular matrices \(B_1\) and \(B_2\) are the connectivity of the internal degrees of freedom and the interface degrees of freedom of the substructures. We explicitly assume that the stiffness matrices of the elements in a substructure are symmetric. Finally, \(C\) represents the connectivity of the interface degrees of freedom and the degrees of freedom of elements not in any substructure. By construction, there is no connection between the internal degrees of freedom of the two substructures. The above matrix can be factorized as follows

\[
\begin{bmatrix}
A_1 & 0 & 0 \\
0 & A_2 & 0 \\
B_1^T & B_2^T & C^*
\end{bmatrix}
\begin{bmatrix}
I & 0 & A_1^{-1}B_1 \\
0 & I & A_2^{-1}B_2 \\
0 & 0 & I
\end{bmatrix}
\]

where \(I\) is the identity matrix and \(C^*\) the so-called Schur complement

\[
C^* = C - B_1^TA_1^{-1}B_1 - B_2^TA_2^{-1}B_2
\]

The terms \(B_1^TA_1^{-1}B_1\) and \(B_2^TA_2^{-1}B_2\) are nothing but the substructures after static condensation of the internal degrees of freedom.

Matrix \(A\) is symmetric positive definite, and first constructs its Cholesky factorization \(LL^T\). Then \(B_1^TA_1^{-1}B_1\) is efficiently computed as follows:

\[
B_1^TA_1^{-1}B_1 = B_1^T(LL^T)^{-1}B_1 = (L^TBL)\]

After all substructures have been assembled, it remains to solve a system of the form

\[
C\cdot u = f
\]

In DIANA the solution of this system can be obtained by factorization of \(C^*-1\), or by using an iterative solver.

This approach to sub-structuring is closely related to domain decomposition. The main difference is that in sub-structuring the elements are partitioned, whereas in domain decomposition the degrees of freedom are partitioned.

A final word must be placed about the effectiveness of sub-structuring. Even if we assume that the cost of assembling the substructures is negligible (they can be reused as is), it is not always true that the factorization of \(C^*\) is cheaper than the factorization of the full matrix. The matrix \(C^*\) is not only smaller but also denser than the full matrix. Especially if the substructures are characterized by a high ratio of interface degrees of freedom to internal degrees of freedom, the use of substructures may be detrimental to the performance. In this case DIANA will give a warning message and turn off sub-structuring.

Footing load on surface – Performance test

In this paragraph some performance measurements for a simple linear test are presented and discussed. The model is a simple block of 100*100*100 m composed of 8-node brick elements (HX24L in DIANA). Linear material properties (\(E = 5\) GPa, \(v = 0.18\)) are chosen for all elements. At one corner of the top-surface of 25*25 m a constant pressure load is applied, leading to an inhomogeneous deformation of the block. Five variations of the model, with different number of elements are analysed.
The bottom surface is supported in vertical direction and to two side-surfaces symmetry-plane boundary conditions have been applied.

The model has been analysed with five different number of elements:

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Number of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model1</td>
<td>20 x 20 x 8 = 3.200</td>
</tr>
<tr>
<td>Model2</td>
<td>40 x 40 x 16 = 25.600</td>
</tr>
<tr>
<td>Model3</td>
<td>60 x 60 x 24 = 84.600</td>
</tr>
<tr>
<td>Model4</td>
<td>90 x 90 x 36 = 291.600</td>
</tr>
<tr>
<td>Model5</td>
<td>100 x 100 x 50 = 500.000</td>
</tr>
</tbody>
</table>

Overview of different models for Footing load on surface test

The analyses have been performed with DIANA 9.3 at a dual Intel® Xeon® X5355 quad-core 2.66 GHz, 24 GB memory, Linux platform.

A break-down of CPU-times and elapsed times is as follows:

<table>
<thead>
<tr>
<th>Model Name</th>
<th>CPU INPUT</th>
<th>CPU MODEL</th>
<th>CPU SOLVE ITERAT</th>
<th>CPU SOLVE PARDIS</th>
<th>CPU POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model1</td>
<td>0.40</td>
<td>0.98</td>
<td>0.49</td>
<td>2.61</td>
<td>0.55</td>
</tr>
<tr>
<td>Model2</td>
<td>3.06</td>
<td>8.75</td>
<td>6.87</td>
<td>38.60</td>
<td>7.65</td>
</tr>
<tr>
<td>Model3</td>
<td>10.21</td>
<td>29.38</td>
<td>32.54</td>
<td>448.46</td>
<td>29.11</td>
</tr>
<tr>
<td>Model4</td>
<td>34.68</td>
<td>100.09</td>
<td>105.94</td>
<td>5081.16</td>
<td>97.81</td>
</tr>
<tr>
<td>Model5</td>
<td>59.74</td>
<td>182.73</td>
<td>258.31</td>
<td>***</td>
<td>174.60</td>
</tr>
</tbody>
</table>

CPU times in seconds of main functions of linear elastic analysis for different model-sizes (** does not fit in memory of computer)

A linear elastic analysis consists of the following operations: INPUT for reading the model-data, MODEL for evaluating the input data, setting up the individual linear stiffness-matrices and definition of the external force-vector, SOLVE for solving the linear set of equation. In this example the CPU-times for the iterative solver as well as for the PARDISO solver are listed. Finally the CPU-times for POST are listed. Post-processing includes calculation of displacements and global stresses and strains in each node of each element and write them to the FEMVIEW result-file.

For the model5 the maximum memory usage is 2.1 Gb for the Iterative solver while for the direct solver uses for the same model 33 Gb.

From the table it can be concluded that the CPU times for INPUT, MODEL and POST increase more or less linear with the number of elements in the model. On the other hand the CPU times for SOLVE ITERAT increase only slightly stronger than linear with the number of elements, whereas the CPU times for SOLVE PARDIS increase almost to the second order of the number of elements of the model.

This simple test illustrates the power of iterative solvers for solving models with a very good numerical condition such as in this test. The numerical condition of a model can be determined by the ratio of the largest and smallest diagonal term in the stiffness matrix. Because in this model all elements have the same size and we only have displacement degrees of freedom, the numerical condition of the stiffness matrix is almost optimum. In following examples we will illustrate that when the numerical condition of the matrix is not so-good or for models for which the bandwidth of the matrix is smaller, e.g. for shell elements which are in general connected to other shell elements in a 2D way, direct solver may be preferred.

From this example we can conclude that for models with majority of solid elements, the iterative solver give superior performance, both with respect to required memory and CPU.

Connection of pipes – Performance test

In this paragraph some performance measurements for a linear analysis of a model of a connection of 2 pipes are presented and discussed. The model is a Y-connection of 2 cylindrical tubes of 4-nodes curved shell elements (Q20SH in DIANA).

Deformed shape of Element Mesh
Linear material properties (E= 210 GPa, v = 0.3) are chosen for all elements. The vertical tube is supported in all directions, whereas at the end of the branch-pipe an axial displacement is defined. Five variations of the model, with different number of elements are analysed.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Number of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shell 10</td>
<td>1.224</td>
</tr>
<tr>
<td>Shell 5</td>
<td>4.833</td>
</tr>
<tr>
<td>Shell 2.5</td>
<td>18.802</td>
</tr>
<tr>
<td>Shell 1.25</td>
<td>74.715</td>
</tr>
<tr>
<td>Shell 0.625</td>
<td>297.929</td>
</tr>
</tbody>
</table>

**Overview of different models for Connection of Pipes test**

The analyses have been performed with DIANA 9.4.3 at a threads.

A break-down of CPU-times and elapsed times is as follows:

<table>
<thead>
<tr>
<th>Model Name</th>
<th>CPU INPUT</th>
<th>CPU MODEL</th>
<th>CPU SOLVE ITERAT</th>
<th>CPU SOLVE PARDISO</th>
<th>CPU POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shell 10</td>
<td>0.10</td>
<td>0.42</td>
<td>0.55</td>
<td>1.28</td>
<td>1.25</td>
</tr>
<tr>
<td>Shell 5</td>
<td>0.33</td>
<td>1.61</td>
<td>4.76</td>
<td>3.58</td>
<td>2.10</td>
</tr>
<tr>
<td>Shell 2.5</td>
<td>1.22</td>
<td>6.29</td>
<td>54.05</td>
<td>10.04</td>
<td>8.03</td>
</tr>
<tr>
<td>Shell 1.25</td>
<td>5.01</td>
<td>25.25</td>
<td>463.22</td>
<td>47.92</td>
<td>31.85</td>
</tr>
<tr>
<td>Shell 0.625</td>
<td>17.10</td>
<td>99.82</td>
<td>2069.32</td>
<td>358.47</td>
<td>135.47</td>
</tr>
</tbody>
</table>

**CPU times in seconds of main functions of linear elastic analysis for different model-sizes.**

A linear elastic analysis consists of the following operations: INPUT for reading the model-data, MODEL for evaluating the input data, setting up the individual linear stiffness-matrices and definition of the external force-vector, SOLVE for solving the linear set of equation. In this example the CPU-times for the iterative solver as well as for the PARDISO solver are listed. Finally the CPU-times for POST are listed. Post-processing includes calculation of displacements and global stresses and strain in each layer for every node of each element and distributed bending moments and cross-section forces in every node of every element and write them to the FEMVIEW result-file.

For the model5 the maximum memory usage is 3.9 Gb for the Iterative solver while for the direct solver uses for the same model 10 Gb. Note that the number of degrees of freedom for this model is 1.7 M, whereas the number of degrees of freedom for the Footing load on surface test is circa 1.6 M. When we compare the CPU times for the iterative and direct solvers for these models they show very different characteristics. Whereas for the model with solid elements (Footing load of surface) the iterative solver gave clearly best performance, for the present model with curved shell elements (Connection of Pipes), the direct solver showed superior performance. The number of degrees of freedom is similar for both models, but the numerical condition and the bandwidth of the stiffness matrix is different. As for the solid element model the bandwidth is large and the numerical condition is good, for the curved shell model, the numerical conditions is not good, but the bandwidth of the matrix is smaller because the elements are connected to other elements mainly as in a 2D model, whereas for the solid model each element is connected to other elements as in a 3D model.

It is in particular the lower bandwidth that can explain the superior behaviour of the direct solver in the curved shell model. In general we can conclude that for models with mainly 2D-type of elements, 2D or shell-structures, the direct solver is the best choice. Especially for curved shell elements the iterative solvers do not show good performance because the degrees of freedom in the system are both related to displacement and rotational degrees of freedom, which each have their own order of stiffness. In such model the ILU pre-conditioner with low threshold for fill-in of the Q-matrix does not lead to efficient convergence of the iterative search process. Therefore, a higher threshold for the matrix fill-in is required, which leads to higher memory usage and more CPU operations in comparison with the direct solver.

**Fly-over – Parallel processing effects**

In this test-case the effect of using 1 or more cores/threads on the analysis times is demonstrated for the PARDISO solver. For this purpose we use a 3D fly-over model that is composed of 118.000 8-nodes quadrilateral curved shell elements (CQ40S). The total number of degrees of freedom is 1.7 M.

Part of finite element mesh of 3D-fly over composed of quadratic curved shell elements.

The analysis has been performed on a dual Intel® Xeon® X5355 quad-core 2.66 GHz, 48 GB memory, Windows Server 2003 x64 platform. The elapsed times for solving of the set of linear
equations by using the PARDISO solver for DIANA 9.4.3 are shown in table below.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>ELAPSED TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 thread</td>
<td>3 min 56 sec</td>
</tr>
<tr>
<td>2 threads</td>
<td>2 min 54 sec</td>
</tr>
<tr>
<td>4 threads</td>
<td>2 min 11 sec</td>
</tr>
<tr>
<td>8 threads</td>
<td>1 min 28 sec</td>
</tr>
</tbody>
</table>

Elapsed times of solver times using PARDISO solver for 3D fly-over model with 1 or more processors

From the table above it can be concluded that the speed-up by using more than one core that can be expected from the PARDISO solver is significant, but it is also clear that the with activating more than 1 thread the effort for distributing the job over the cores increases.

Footing load on surface – Parallel processing effects

The finite element model is the same as the 500,000 element model described above in the paragraph with the same name.

The analysis has been performed with DIANA 9.4.3 on a dual Intel® Xeon® X5550 quad-core 2.66GHz, 48 GB memory, Linux platform. The elapsed times for solving of the set of linear equations by using the Iterative solver are show in the tables below.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>ELAPSED TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 thread</td>
<td>28 sec</td>
</tr>
<tr>
<td>2 threads</td>
<td>17 sec</td>
</tr>
<tr>
<td>4 threads</td>
<td>16 sec</td>
</tr>
<tr>
<td>8 threads</td>
<td>15 sec</td>
</tr>
</tbody>
</table>

Elapsed times of solver times using Iterative solver for 500,000 element Footing load on surface model with 1 or more threads, for the first time in non-linear analysis.

Solver times are listed for the first time the solver is called and also for the second time in a non-linear analysis. For the second time the data-required for distribution of the analysis job over the different cores has been kept in memory from the first time and therefore time to built-up these data in the solver memory can be saved. Another reason why the second solver call required less time than the first one is because the pre-conditioned matrix \( Q \) from the first step is reused in the second step.

The speed-up from 1 to 8 threads for this model is not very impressive because the box model leads to relatively large interfaces between the different sub-domains. Similar analysis of two other models will be shown in the next paragraphs to illustrate that element connectivity may have a large impact on the speed-up in using more than 1 thread.

GEOMEC model – Parallel processing effects

This is a typical geomechanical analysis model of an oil/gas reservoir. The model contains different rock-formations, which are described with 4-node tetrahedron elements (TE12L) and sliding surfaces which are modelled with triangular interface elements (T18IF). The model has 156,000 nodes, circa 400,000 degrees of freedom, 800,000 TE12L elements and 40,000 T18IF elements. A dead-weight-load and variation of gas-pressure load are applied. In this model we use the iterative solver.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>ELAPSED TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 thread</td>
<td>146 sec</td>
</tr>
<tr>
<td>2 threads</td>
<td>99 sec</td>
</tr>
<tr>
<td>4 threads</td>
<td>73 sec</td>
</tr>
<tr>
<td>8 threads</td>
<td>45 sec</td>
</tr>
</tbody>
</table>

Elapsed times of solver times using Iterative solver for GEOMEC model with 1 or more threads, for the first time in non-linear analysis.
Elapsed times of solver times using Iterative solver for GEOMEC model with 1 or more threads, for the second time in non-linear analysis.

The elapsed times listed in the tables above make clear that in a nonlinear analysis, activating more than one processor does not always lead to shortened analysis times, because the overhead of distributing the job over the cores can be expensive, especially for models for which the interfaces between the different sub-domains are relatively large.

Concrete Beam – Parallel processing effects

This model is a reinforced concrete beam that is composed of 12,600 second order 20-nodes brick elements (CHX60) and reinforcement bars in the lower part of the beam and a for a series of stir-ups.

The beam is supported at both ends and a point-load is applied in vertical direction in the middle of the top-surface.

Finite element mesh of concrete beam with reinforcements.

The analysis has been performed with DIANA 9.4.3 on a dual Intel® Xeon® X5550 quad-core 2.66GHz, 48 GB memory, Linux platform. The elapsed times for solving of the set of linear equations by using the Iterative solver are show in the tables below.

Elapsed times of solver times using Iterative solver for Concrete beam model with 1 or more threads, for the first time in non-linear analysis.

The elapsed times for the concrete beam show a significant better speed-up for using more than 1 thread than for the GEOMEC or Footing load on surface models. The better result is related to the element connectivity of the beam. Because of the elongated shape of the beam, sub-domains can be defined over the cross-section of the beam, which are relatively small to the number of nodes in the model. These small interfaces require less overhead for distributing the analysis work over the different processors than for the more lock shaped models presented before. For optimum speed-up in parallel-processing the interfaces between the different sub-domains should be small.