

# State-of-the-art distributed parallel computational techniques in industrial finite element analysis

**SIEMENS**

## White Paper

This white paper reviews distributed parallel computational techniques used in modern industrial finite element analysis. The conceptual aspects as well as the basic mathematics of the techniques are reviewed. Application examples from real-life industrial applications illustrate the techniques.

# Contents

- Introduction.....3
- Industrial finite element analysis .....4
- Geometry domain decomposition.....5
- Recursive normal modes analysis .....7
- Computational kernel technology.....12
- Industrial application examples.....15
- Conclusions .....18
- References.....19

## Introduction

The high fidelity requirements of industrial analyses result in ever increasing finite element model sizes. The related computational solutions demand computer resources far in excess of a single engineering workstation. This fact necessitates the use of state-of-the-art computational techniques that combine the resources of multiple workstations and exploit their internal parallelism.

Such techniques are comprised of three major components. First, various domain decomposition techniques are applied to subdivide very large problems into smaller, albeit not necessarily complete, problems. Second, special numerical solution methods are needed to solve various engineering analysis problems in a distributed form. Finally, efficient and hardware-intelligent computational kernels are the key to exploiting the internal parallel capabilities of the individual workstations.

# Industrial finite element analysis

Industrial finite element analysis is a major tool in product lifecycle management. Its historical focus has always been on the operational phase of product lifecycle; however, today the operational scenarios traverse a much wider range of physical phenomena. These require the simulation of various external multi-physics effects on the product and its interaction with other components in a multi-body environment. There are examples of these in life; the behavior of a car body traveling over a rough road in the auto industry, or the opening of a landing gear of an aircraft in the airplane industry.

The product manufacturing phase of the lifecycle has also become the subject of simulation via finite element technology since some products like bridges, wind-mill towers or oil drilling platforms are not stable during the construction process. Even the maintenance and disposal phases of a product's lifecycle are now analyzed with finite element technology. Assessing the damaged physical integrity of a structure is of utmost importance in the aerospace and defense industries.

Industrial finite element analyses are executed in the time and in the frequency domain; the structures may be excited by external forces or freely vibrating. A most important case of industrial analysis is the latter, free vibration of undamped structures, or normal modes analysis.

In this analysis case, there is no velocity-dependent damping effect and no external loads on the structure. The solution of this problem is not just important in its own right as the means of establishing a structure's natural frequencies, but also as the fundamental component of the class of so-called modal solutions.

We choose to discuss the state-of-the-art in distributed technologies, subject of our focus in connection with this computation not just because of its above stated importance, but its conceptual simplicity. The technology carries over into the more difficult analysis scenarios, but their description is much less educational.

There are two major challenges in industrial finite element analysis today. The first is that matrices of industrial problems are of the order of several million to tens of millions of degrees of freedom. This is mainly due to the significant advancement of the geometric modeling tools. The second challenge lies in the changing computer hardware environment. The single, massive supercomputers of the past century have been replaced by a collection of many, high performance computers.

These challenges are answered by the techniques described in the following three sections: subdividing the large problems, distributing the solutions and exploiting performance of the individual computer nodes.

## Geometry domain decomposition

The role of this step is to partition an engineering problem to decrease processing resource requirements while minimizing the size of common boundaries to reduce interprocess communication. The geometry domain decomposition technique may be applied directly to a finite element graph or the graph of finite element matrices. The most commonly used methods are based on recursive graph partitioning using spectral bisection.

The spectral bisection method [7.] is based on finding minimal cuts in a graph. The method is building the Laplacian matrix of the graph and finding its second eigenvector, the so-called Fiedler vector, described in [2.]. Let us consider a simple graph shown in Figure 1.

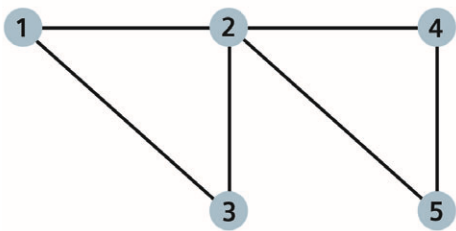


Figure 1: Example graph.

The Laplacian matrix expresses the connectivity of graphs by having as many rows as vertices are in the graph and having the degree of the vertex (the number of edges connected to it) on the diagonal term of the row. The other terms of the row corresponding to a vertex contain -1 if the vertex indexed by the term is connected to the pivot vertex, zero otherwise. The Laplacian matrix for the above example is:

$$1. \quad L = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 \\ -1 & -1 & 2 & 0 & 0 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & -1 & 0 & -1 & 2 \end{bmatrix}$$

The second eigenvalue and eigenvector of this matrix are

$$2. \quad L\varphi_2 = \lambda_2\varphi_2$$

or

$$3. \quad \begin{bmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 \\ -1 & -1 & 2 & 0 & 0 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & -1 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} -1/2 \\ 0 \\ -1/2 \\ 1/2 \\ 1/2 \end{bmatrix} = 1 \begin{bmatrix} -1/2 \\ 0 \\ -1/2 \\ 1/2 \\ 1/2 \end{bmatrix}$$

The minimal cut of the graph is along the vertex that corresponds to the smallest absolute value term of the Fiedler vector. In our example this is at vertex 2. Hence the graph is partitioned as shown in Figure 2.

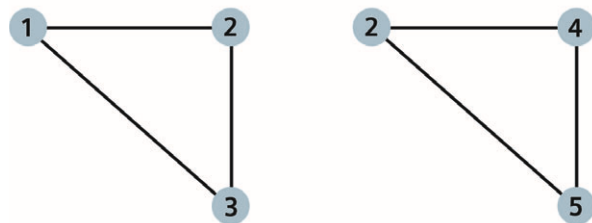


Figure 2: Partitioned graph.

The technology applies to finite element meshes as well, utilizing the analogy between the mesh and a regular graph. The nodes of the mesh are the vertices of the graph and the edges of the graph are represented by the element connections. Let us consider a very simple finite element model with 16 plate elements and 25 nodes. Only those nodes that are important for the proceeding discussion are numbered in Figure 3.

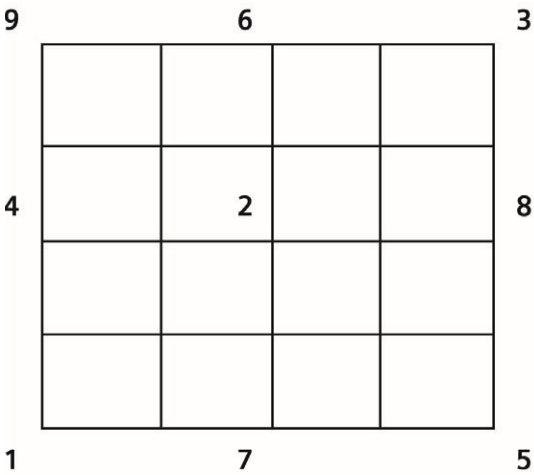


Figure 3: Example finite element mesh.

The partitioning of the graph is executed along a minimal vertex cut that does not cut through any of the elements. In our example this is rather simple. Finding such cuts in practical implementations involves some heuristics, since even in our simple example two obvious and equally minimal cuts (one horizontal and one vertical) exist.

Choosing the vertical cut, the two partitions are shown in Figure 4.

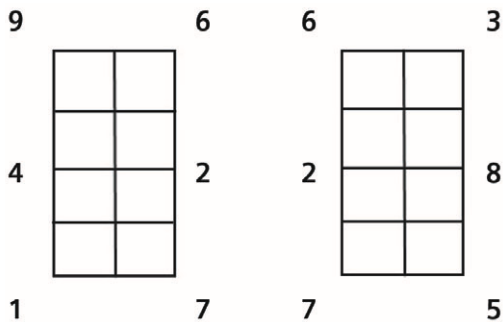


Figure 4: First partitioning of mesh.

Further partitioning of the partitions follows recursively. The result of this is the four components shown in Figure 5, where the interiors of the components were omitted and only the boundaries are shown. If that is the desired number, or the sizes of the components are sufficiently small, the process stops.

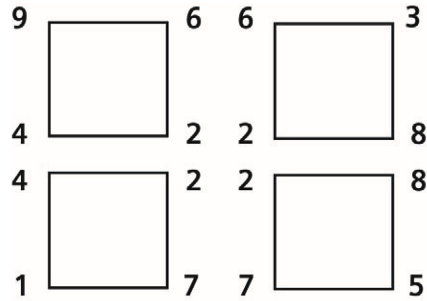


Figure 5: Recursive partitioning.

The physical solution computations in the geometry partitions are dependent through their common boundaries. Hence care must be taken to minimize the boundary sizes between partitions in order to minimize communication costs.

Those considerations are out of our focus here. There are several publicly available domain decomposition tools available, one is described in [3.]. Note that automated geometry domain partitioning techniques usually provide only even numbers of domains due to binary graph partitioning. This is, however, not a significant restriction as the distributed memory hardware environments tend to have even number of processors in most cases.









where  $j$  is either 3 or 6. The final problem is the same size as the number of columns in  $S$ , i.e.  $(d)$ , and it has the structure of a main diagonal with some coupling matrices.

The distributed execution of the operations is now quite clear. All the interior component calculations in the two stages, the static condensation and dynamic reduction, may be executed independently of each other on different nodes of the distributed hardware. This is followed by the solution of the reduced order problem on a single (preferably the master) node.

The eigenvalues of the reduced order problem are good approximations of the eigenvalues of the original problem,

$$39. \quad \overline{\lambda} \approx \lambda$$

Hence no further action is necessary. On the other hand, a back transformation of the eigenvectors are required to redo the effects reduction process. The example model's reduced order eigenvector is partitioned as

$$40. \quad \phi_d = \begin{bmatrix} \phi_q^1 \\ \phi_q^2 \\ \phi_q^3 \\ \phi_q^4 \\ \phi_q^5 \\ \phi_q^6 \\ \phi_q^7 \end{bmatrix}$$

First, the dynamic reduction is recovered for the interiors

$$41. \quad \overline{\phi}_t^i = \Phi_{oq_o}^T \phi_q^i, \quad i = 1, 2, 4, 5$$

and for the boundaries

$$42. \quad \phi_t^j = \Phi_{iq_i}^T \phi_q^j, \quad j = 3, 6, 7$$

Secondly, the effects of the static condensation are accounted for as

$$43. \quad \phi_o^i = \overline{\phi}_o^i + G_{ot}^{i,j} \phi_t^j, \quad i = 1, 2, 4, 5$$

The boundary components  $\phi_t^j$  did not change during the static condensation. For more details on the numerical aspects and practical considerations of the technique see [5.].

## Computational kernel technology

There are three major computational components in the above technique: the solution of the local eigenvalue problems, the factorization of the matrices and matrix-matrix multiplications.

The solution of the individual, local eigenvalue problems is done predominantly by the Lanczos method in all commercial finite element analysis tools. This topic is also outside of our focus here. The description of the industrial method is in [4.].

The local matrix factorization is very important because it is also part of the eigenvalue solution. Hence we will focus on that topic in this section. The recursive operations described in the past section frequently used the equation

$$44. \quad G_{ot}^{i,j} = -K_{oo}^{-1,i} K_{ot}^{i,j}$$

The posted inverse is, however, never computed explicitly. Instead a matrix factorization

$$45. \quad K_{oo}^i = L_{oo}^i D_{oo}^i L_{oo}^{T,i}$$

followed by a forward-backward substitution

$$46. \quad (L_{oo}^i D_{oo}^i L_{oo}^{T,i}) G_{ot}^{i,j} = -K_{ot}^{i,j}$$

is used to carry out the operation.

The premier, industry standard factorization technique in finite element analysis is the multi-frontal technique [1.]. The simplified idea of the method is to execute the sparse factorization in terms of dense submatrices, the "frontal matrices". The frontal method is really an implementation of Gaussian elimination by eliminating several rows at the same time.

The multi-frontal method is an extension of this by recognizing that many separate fronts can be eliminated simultaneously by following the elimination pattern of some reordering, like the minimum

degree method. Hence, the method starts with a symbolic factorization-based reordering:

$$47. \quad PK_{oo}^i P^T = A_1$$

This is a symbolically reordered matrix, meaning it is not rewritten in the new order: it is only accessed in the new order. This order is encapsulated in a graph, called the elimination tree. This tree also visualizes the columns that may be eliminated simultaneously, pinpointing the opportunities for parallel execution. An example of the elimination tree of a matrix is shown in Figure 7.

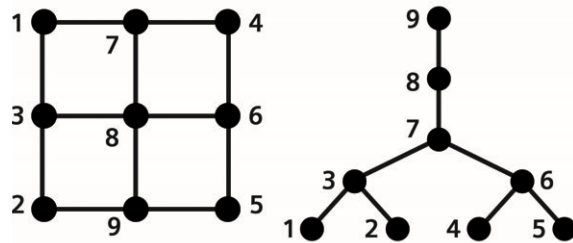


Figure 7: Matrix and its elimination tree.

The goal of the reordering is ultimately to produce an efficient numerical factorization with respect to the number of floating point operations and the final size of the factor matrix. The latter significantly influences the performance of the forward-backward substitution operation usually following the factorization.

The factorization is executed in blocks of the columns of the matrix, the so-called supernodes. A supernode is a set of neighboring columns whose nonzero pattern is identical outside of the diagonal block, with the size of the number of columns in the supernode. The diagonal block is considered to be dense, and since outside the block the columns have the same sparsity pattern, the whole block could be gathered into a dense rectangular matrix.

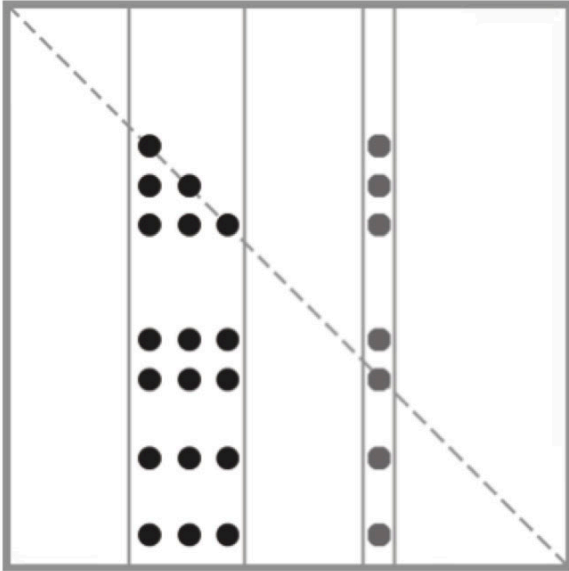


Figure 8: Matrix supernode definition.

An example of the supernode of a matrix is shown in Figure 8. The three columns bracketed by the vertical lines is a supernode. The dots represent the identical nonzero pattern in the columns. Since the definition requires the columns to be neighbors, another column with the same sparsity pattern, shown on the right, does not belong to the supernode.

Let  $A_j$  be partitioned (symbolically again) to isolate the first supernode and also assume that the inverse of the  $s_j \times s_j$  diagonal block  $D_j$  exists. Then

$$48. \quad A_j = \begin{bmatrix} D_1 & S_1^T \\ S_1 & B_1 \end{bmatrix}$$

The first factorization step is

$$49. \quad A_j = \begin{bmatrix} I_1 & 0 \\ S_1 D_1^{-1} & I_{n-1} \end{bmatrix} \begin{bmatrix} D_1 & 0 \\ 0 & B_1 - S_1 D_1^{-1} S_1^T \end{bmatrix} \begin{bmatrix} I_1 & D_1^{-1} S_1^T \\ 0 & I_{n-1} \end{bmatrix}$$

Here the subscript for  $I_1$  indicates the solution step rather than the size of the identity matrix. The size is  $s_1$  as indicated by the size of  $S_1$ . The process is repeated by taking

$$50. \quad A_2 = B_1 - S_1 D_1^{-1} S_1^T$$

We partition the next supernode as

$$51. \quad A_2 = \begin{bmatrix} D_2 & S_2^T \\ S_2 & B_2 \end{bmatrix}$$

and factorizing as above. The process repeats until all the supernodes, for example  $k$ , are eliminated. The final factors are built as

$$52. \quad L = \begin{bmatrix} I_1 & 0 & 0 & \cdot & 0 \\ S_1 D_1^{-1} & I_2 & 0 & \cdot & 0 \\ | & S_2 D_2^{-1} & I_3 & \cdot & 0 \\ | & | & \cdot & \cdot & \cdot \\ | & | & & \cdot & I_k \end{bmatrix}$$

and

$$53. \quad D = \begin{bmatrix} D_1 & 0 & 0 & 0 & 0 \\ 0 & D_2 & 0 & 0 & 0 \\ 0 & 0 & D_3 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 0 & B_k - S_k D_k^{-1} S_k^T \end{bmatrix}$$

Here  $D$  is built from variable size  $s_i \times s_i$  diagonal blocks of the supernodes, the  $D_i$  matrices. The  $S_i D_i^{-1}$  sub-matrices are rectangular, extending to the bottom of the factor. The process stops when  $s_k$  is desirably small.

It is evident from the form above that the main computational component, the kernel of the factorization process, is the repeated step

$$54. \quad B_i - S_i D_i^{-1} S_i^T$$

Since the  $B_i$  matrix is ever changing, gradually smaller and constantly updated partition of the original  $A_i$  matrix, this step is called a matrix update. The matrix update is commonly executed between the supernode and same-size rectangular partitions of the submatrix to be updated, called panels.

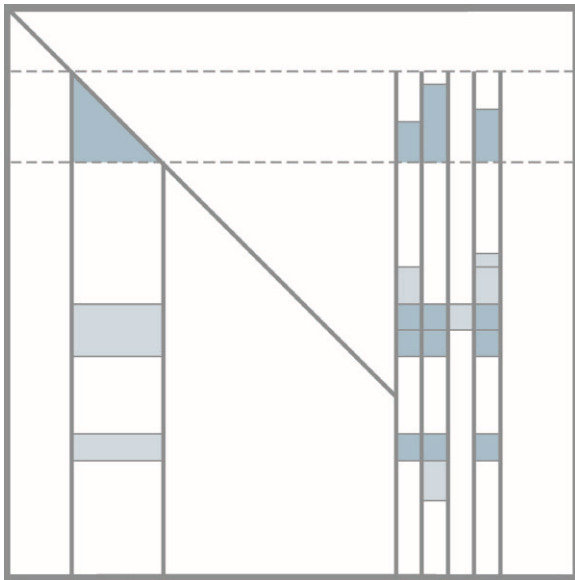


Figure 9: Supernodal update operation.

An example of the supernodal update operation is shown in Figure 9. The  $S_i$  supernode is the block on the left and the panel, a part of the  $B_i$  matrix to be updated, is on the right. The dotted areas of the supernode represent the nonzeros. The panel on the right has a different sparsity pattern shown by dotted areas not matching those of the supernode. The darker areas marked by the slant lines are the areas updated by the supernode.

If the size of this matrix is such that the current supernode and a panel both fit into cache memory of the computer for fast operation, the operation is uniquely suited to be executed by multicore processors. Practical implementations of this operation in various libraries aim at such an efficient arrangement. Performance results on various modern hardware platforms are presented in [6.].

The other computationally intensive kernel component is the execution of matrix multiplication, mostly by the mass matrix. The mass matrix is very sparse by the nature of the finite element technology, but still the sheer size of the problem and the denseness of the vectors that are being multiplied require special care. While it is conceptually a simple operation, the details of mapping such operations to the various multicore hardware platforms are far from trivial, as shown in [8.].

Even though multicore processors usually come in even number arrangements, the computational kernel problem sizes (the number of columns in a supernode for example) cannot be assured to have even components. Practical implementations of the multicore computational operations mentioned above can efficiently cope with such cases.

## Industrial application examples

To analyze the effect of the above technologies, two industrial analysis jobs executed with NX™ Nastran® software are discussed. NX Nastran spearheaded some of the computational techniques discussed above and has efficient implementations on a variety of hardware platforms. Most engineering solution computations can take advantage of the distributed technologies.

Let us first consider the analysis of a trimmed car body automobile model. Such models have all major components of the car structure, chassis, windows, doors, etc. incorporated. An example is shown in Figure 10.

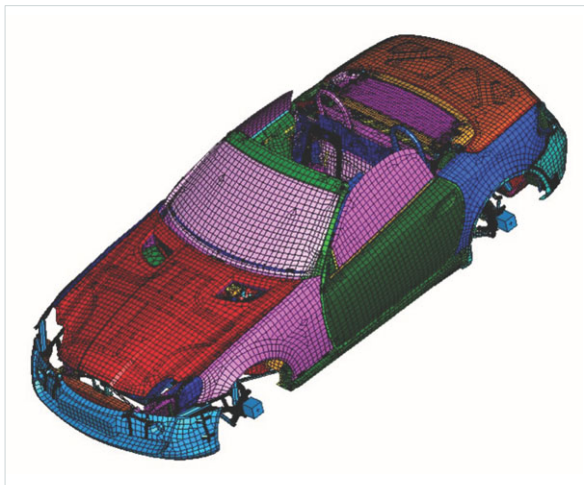


Figure 10: Car body model.

The example model had 1.3 million node points and approximately 1.2 million elements, mainly shell. The resulting matrix size after the elimination of constraints was 7.9 million degrees of freedom. Normal modes analysis was executed for various frequency ranges; for example, the range of 0 to 200 Hz contained about 1000 normal modes. The model was decomposed into 512 partitions.

The task of finding the natural frequencies and corresponding mode shapes of such a model is truly an enormous one. It is an overnight job with more than a terabyte of I/O operations. The execution on a single processor is impractical, considering the work environment and time schedule at automobile companies.

The IBM P5-575 hardware environment we used to execute the analysis had 111 nodes with 4 dual core POWER 5 processors with 1.9 GHz speed per node, totaling 888 processors. The machine had 3.5 TB aggregate memory and 100 TB total disk space. The nodes were connected with IBM's high performance switch (HPS) providing 2 GB/sec bidirectional bandwidth. The operating software environment was AIX Version 5.3 with PE V4.2.

The results of this analysis job are presented in Table 1. The first row represents a one shot (no partitioning) serial execution. That run was impossible to execute on a single node, hence its elapsed time was estimated from partial runs. The rest of the rows are all complete executions of the same analysis, and the normal modes computation elapsed times are reported in minutes:seconds.

Nodes	Time	Speedup
0	7,000:00	1.00
1	1,719:26	4.00
2	877:58	7.8
4	235:01	29.3
8	139:52	49.2
16	88:44	77.5
32	71:18	96.5
64	66:03	104.1
128	64:56	105.9

Table 1: Execution statistics of trimmed car body model.

The table shows about a 100-fold speed-up over the serial execution using 128 nodes of the above environment. The elapsed time of the last line is less than three hours, making this technology a tool that enables the engineer to execute such analysis during the daily work flow.

Another interesting aspect of this enabling technology was found by extending the frequency range of the analysis, which was 200 Hz in the above. Table 2 shows normalized results of varying the range from 100 to 500 Hz.

Range	Modes	Time
100	1.0	1.0
200	2.41	1.08
300	4.67	1.21
400	7.44	1.34
500	10.93	1.55

Table 2: Advantage of technology with increased frequency range for the car body model.

When the frequency range was doubled from 100 to 200 Hz, the number of computed modes was 2.41 times more, but increase in the analysis time was only 8 percent. The range of 500 Hz resulted in 10.93 times more number of modes found for only a 55 percent increase in computational cost. This is due to the fact that the cost of partitioning and reduction is a dominant component of this work, and that does not change when extending the frequency range.

We also consider a different class of example, that of an engine block shown in Figure 11.

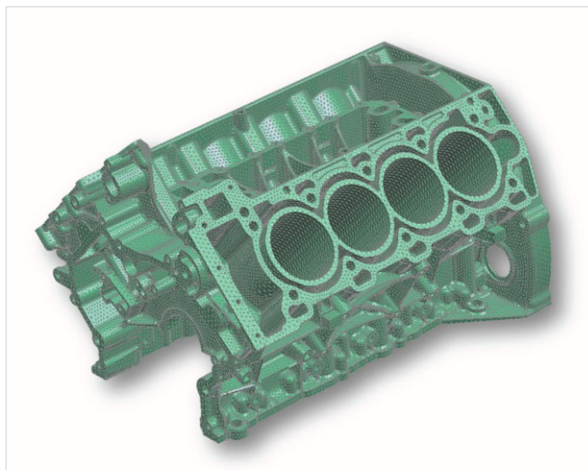


Figure 11: Engine block.

This model had approximately 3.6 million node points, mainly due to the richness of the geometric details. This is almost three times the number of node points of the car body model. The model, however, was discretized by 2.3 million solid (in our case tetrahedral) elements, that carry a lower computational complexity. The resulting degrees of freedom was still large, 10.8 million. By the nature of such a model, the frequency range of interest of the engineer is also wider, reaching into the tens of thousands of Hz. This model was decomposed into 256 partitions.

Nodes	Time	Speedup
0	17,500:00	1.00
1	4,370:52	4.00
2	2,548:23	7.11
4	1,402:05	12.47
8	1,019:13	17.15
16	678:09	25.78
32	505:36	34.58
64	354:53	49.27

Table 3: Execution statistics of engine block model.

Table 3 presents the statistics of the above example analyzed on a 64-node cluster of 1.85 GHz dual core Intel Xeon CPUs. The machine had 4 gigabytes of memory and 200 gigabytes of local disk space per node. The nodes were connected with gigabit ethernet interconnect. The operational software was SUSE Linux and HP-MPI.

The efficiency of this analysis is lower than the car body example, due to the higher cost of domain decomposition and the higher boundary sizes, both inevitable consequences of the type of model. Still the 50-fold improvement on 64 nodes is a spectacular result.

The widening frequency range experiment shown in Table 4 shows normalized results of extending the range from 10,000 to 50,000 Hz.

Range	Modes	Time
10,000	1.0	1.0
20,000	2.95	1.25
30,000	5.61	1.28
40,000	8.79	1.32
50,000	12.57	1.34

Table 4: Advantage of technology with increased frequency range for the engine block model.

The increase in time when widening the frequency range is even less in this case. Widening the range five-fold resulted in 12.57 times more normal modes, but only 34 percent increase in computational cost. The explanation is that the cost of the partitioning and reduction is more dominant in this case, because of the bulky and highly connected nature of the geometry. The partitioning component of the work, as explained in the connection with the car body, does not change. It would of course change when varying the number of partitions with wider ranges. Such a dynamic adjustment of this technology is the subject of our further research.

## Conclusions

We found that clusters of workstations with multicore processors connected by a proper network provide an environment enabling the solution of very large industrial finite element analysis problems. The solution requires significant investment in the application software to efficiently utilize such environments.

We also showed that the normal modes problem of industrial finite analysis may be restructured into a recursively partitioned and distributed solution technique. The technique, coupled with computational kernels executing linear algebraic operations on multicore processors, produced spectacular results in various auto industry applications. The technology is not limited to a specific industry; similar applications in other industries also produce excellent results.

It is a practical desire to execute above industrial finite element computations in a grid computing

environment. There are, however, several obstacles in the way. One is that our computations require an organized node-to-node communication. The second obstacle is that the local processors and storage devices need a high-speed, wide bandwidth connection; otherwise the interprocessor data exchange slows down. Third, similar computational complexity components are expected to complete at the same time in the distributed scheme above.

The common grid computing environment combining computer resources from many administrative domains do not directly support these needs. Furthermore, the usually nonhomogenous grid environments cause synchronization problems that may be hard to overcome. Nevertheless, it is theoretically feasible to extend our distributed solutions to such an environment.

## References

1. Duff, I. S. and Reid, J. K., "The multi-frontal solution of indefinite sparse linear systems," ACM ToMS, Vol. 9, pp. 302-325, 1983.
2. Fiedler, M., "Laplacian of graphs and algebraic connectivity," Combinatorics and graph theory, Vol. 25, pp. 57-70, 1989.
3. Karypis, G. and Kumar, V., Williams, S., "ParMETIS: Parallel graph partitioning and sparse matrix library," University of Minnesota, 1998.
4. Komzsik, L. "The Lanczos method: Evolution and application," SIAM, Philadelphia, 2003.
5. Komzsik, L. "Computational techniques of finite element analysis, 2nd edition," Taylor and Francis, 2007.
6. Li, X., "Evaluation of sparse LU factorization and triangular solution on multicore architectures", VECPAR, Toulouse, 2008.
7. Pothen, A., Simon, H. and Liou, K.P., "Partitioning sparse matrices with eigenvectors of graphs," SIAM Journal of Matrix Analysis and Applications, Vol. 11, pp. 430-452, 1990.
8. Williams, S. et al, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," ACM transactions, Reno, 2007.

The original version of this article appeared in Trends Parallel, Distributed, Grid and Cloud Computing, as Chapter 1 by Louis Komzsik, Saxe-Coburg Publications, UK, 2011.

## About Siemens PLM Software

Siemens PLM Software, a business unit of the Siemens Industry Automation Division, is a leading global provider of product lifecycle management (PLM) software and services with 6.7 million licensed seats and more than 69,500 customers worldwide. Headquartered in Plano, Texas, Siemens PLM Software works collaboratively with companies to deliver open solutions that help them turn more ideas into successful products. For more information on Siemens PLM Software products and services, visit [www.siemens.com/plm](http://www.siemens.com/plm).

### [www.siemens.com/plm](http://www.siemens.com/plm)

All rights reserved. Siemens and the Siemens logo are registered trademarks of Siemens AG. D-Cubed, Femap, Geolus, GO PLM, I-deas, Insight, JT, NX, Parasolid, Solid Edge, Teamcenter, Tecnomatix and Velocity Series are trademarks or registered trademarks of Siemens Product Lifecycle Management Software Inc. or its subsidiaries in the United States and in other countries. Nastran is a registered trademark of the National Aeronautics and Space Administration. All other logos, trademarks, registered trademarks or service marks used herein are the property of their respective holders.

© 2011 Siemens Product Lifecycle Management Software Inc.

X16 25721 9/11 C

## Siemens Industry Software

### Headquarters

Granite Park One  
5800 Granite Parkway  
Suite 600  
Plano, TX 75024  
USA  
+1 972 987 3000  
Fax +1 972 987 3398

### Americas

Granite Park One  
5800 Granite Parkway  
Suite 600  
Plano, TX 75024  
USA  
+1 800 498 5351  
Fax +1 972 987 3398

### Europe

3 Knoll Road  
Camberley  
Surrey GU15 3SY  
United Kingdom  
+44 (0) 1276 702000  
Fax +44 (0) 1276 702130

### Asia-Pacific

Suites 6804-8, 68/F  
Central Plaza  
18 Harbour Road  
WanChai  
Hong Kong  
+852 2230 3333  
Fax +852 2230 3210