

Utilization of Parallel Solver Libraries to solve Structural and Fluid problems

Anil Kumbhar¹, Kiran Chakravarthy¹, Ramdass Keshavamurthy², G V Rao¹

¹ CAE R&D Services, Cranes Software International Limited, 5th Floor, Block-I, Shankaranarayana Bldg., MG Road, Bangalore, 560 001, INDIA; ² Corresponding author: ramdass@cranesoftware.com

ABSTRACT

“With the processors becoming MULTICORE, even desktop operations are becoming increasingly parallel. DUAL core computers are becoming the norm and QUAD Core is very affordable for even Small to Medium Enterprises (SME). FEA being commonly used for solving large size problems (millions of Degrees of Freedom) even on desktops, there is a need for FEA Vendors to upgrade the solvers to exploit the multi-core capabilities. If one were to develop and maintain these solutions, it would involve considerable cost and time. However, some methodologies like OpenMP that were used for multi processor shared memory computers are equally valid for the MULTICORE systems. It is in this context that the Parallel Math and Matrix libraries provided by the hardware vendors like INTEL, which are fine tuned for performance on their processors and memory configurations, offer a significant advantage. It is profitable to exploit the availability of these libraries to cut short the development time and cost. This approach would allow the FEA Vendors to quickly adapt and exploit the newer hardware and the associated technologies. In this paper we discuss our experience of using the INTEL MATH KERNEL LIBRARY for extending NISA Solvers on to multi core systems. We would also define a framework for the users to incorporate the hardware vendor specific solvers for best performance”.

Keywords: Finite element method, Parallel computing, OpenMP, Solver , NISA

1. INTRODUCTION

The finite element method (FEM) [1] has been established over years as a versatile technique because of its flexibility to handle complex structural problems effectively. It is being used for solving fluid, heat and electromagnetic problems as well. FEM method finally results in a set of algebraic equations, which need to be solved numerically. With the desktop computers becoming more powerful, and people using automated meshing tools, it is not uncommon to come across models with a few million degrees of freedom (DOF). This is particularly true for 3-D complex geometries. In order to have reasonable turn-around time to solve FEM models, it requires availability of high-performance computer applications that can exploit the capabilities of the hardware. One way this can be achieved is by using solvers provided/recommended by the hardware vendors which are fine tuned for optimal performance. These libraries involve BLAS, LAPACK, sparse and other solvers. In order to get advantage of these libraries, the user needs to get the entire application compiled every time and on each platform, which is practically difficult and sometimes impossible. Hence a solver plug-in approach will minimize the effort by allowing the user to change/update the plug-in without affecting the main application.

2. PROBLEM STATMENT

In the context of FEM, there are broadly two types of solvers- Direct and Iterative solver [1], with each method having its own advantages and disadvantages. In case of iterative method convergence is a critical issue as it might require pre-conditioners, which are problem specific, to obtain or accelerate convergence. In case of direct method, convergence is more predictable. However it might take longer time and require huge memory to hold factored matrix.

Existing NISA software, based on FEM, supports non parallel sparse and iterative solvers in addition to a frontal solver. Since hardware constantly evolves , one should be able to get best performance on every hardware configuration , with minimum change in the application code. This is achievable with vendor specific libraries. Towards this, plug-in development frame work is proposed with which a vendor library can be easily plugged in to NISA. In addition, the user can add his own libraries. In future we plan to incorporate many vendor specific libraries to offer a greater choice to users. Our use of MKL is a first step in that direction.

3. IMPLEMENTATION

In this section the focus is on implementation issues surrounding the development of a plug-in facility in NISA.

3.1 Interface Requirement

Given the wide diversities in the requirement of solver libraries, it may not be possible to provide a comprehensive interface[2,3,4]. Hence in this implementation we have restricted ourselves to providing an interface limited to symmetric/unsymmetric sparse solvers.

General steps involved in Direct Sparse Methods are 1) Sparseness computation of system stiffness matrix followed by reordering and symbolic factorization 2) Factorization 3) Forward and backward substitution.

In linear static analysis of a structural system, for example, the steps involved are straightforward. However, in the case of a non-linear analysis, steps 2 and 3 are repeatedly used.

3.2 Existing Solver Libraries

There are lots of commercial and free software solver libraries which are parallel based on OpenMP and/or MPI.

Some of vender supplied solvers are PASDISO and DSS from Intel which comes with Math kernel Library [5], Watson Sparse Matrix Package (WSMP) from IBM [6,7], CGSSFS from Sun Performance Library and SGI solver.

All of above discussed solvers are available for commercial purposes. But there are a few solvers available free and equally efficient and capable, for solving large systems. Some of these are MULTifrontal Massively Parallel sparse direct Solver (MUMPS) [8] and SuperLU [9] and so on.

These solvers are available in different parallelization modes. Some include only thread level parallelization and some are extended to cluster of SMP nodes, using MPI for Inter processor communication and threads within processors, resulting in best use of available computational resources. In present interface, the focus is on use of thread level parallelization only.

Thus in the present study PARDISO Solver available in Intel Math Kernel Library is made use of. PARDISO solver is available in parallel form with OpenMP threads.

3.3 Matrix Storage

Matrix generated from Finite element model is inherently sparse and typically sparseness varies in the range of 2 to 5% and only non-zero entries are stored. Generally used formats for storage are coordinate format (COO), compressed sparse row format (CSR), compressed sparse column format (CSC) [10]. Linear solver packages widely use CSR format which is easy and memory efficient for storing matrices.

3.4 Algorithm

In NISA, the choice of solver is through an executive command in the input file. This feature is extended to provide an option for USER defined solver in the following form,

SOLVER=USER, <NAME>.

The keyword NAME here refers to the solver NAME in the plug-in. For e.g., PARDISO in the case of MKL.

Typical flow of NISA for linear analysis is shown in Figure 1 along with solver plug-in. The user is expected to provide all the required input and output parameters for the Solver in the plug-in. Additional solver specific implementation parameters can be kept in solver plug-in.

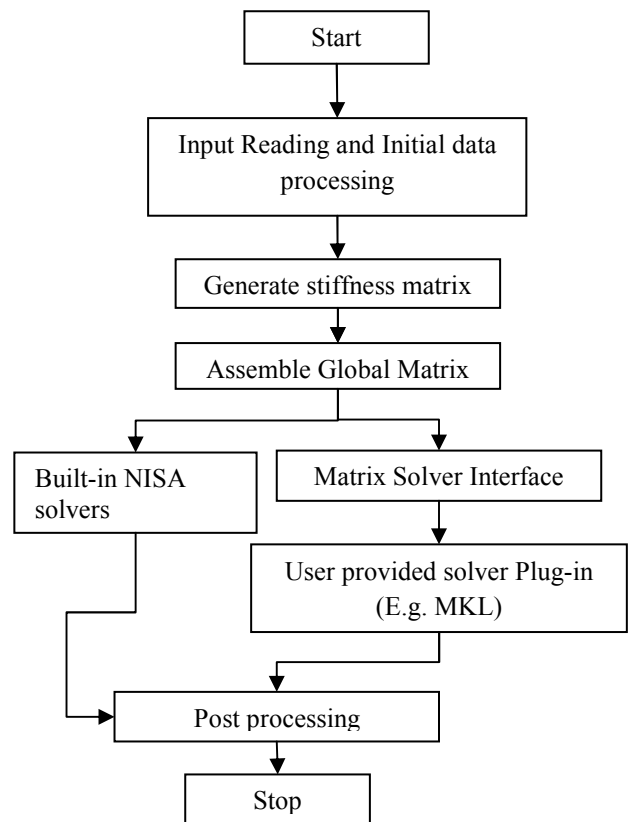


Figure 1 Flow diagram for NISA with solver Interface

4. TEST CASES

The studies were conducted on a dual core and a quad core with 2 processors (resulting in 8 cores) hardware configurations. The study includes both structural and fluid flow analyses that involve symmetric and unsymmetrical matrices, respectively. In this paper

results for eight core processor (2 X 4) are presented, the configuration of which is given in Table 1.

Table 1 Hardware configuration

	Intel Quad Core
Processor	Intel Xeon X5355
No Of Processors	2
Processor Clock Speed	2.66 GHz
RAM	16 GB
Swap space (Virtual Memory)	16 GB
Operating System	Window XP Professional X64

The study is focused on the accuracy, speed and scalability of solver.

The structural analysis was performed for a manifold, used in an automobile engine subject to internal pressure. The manifold is meshed using 8,72,000 tetrahedral elements resulting in 6,94,000 degrees of freedom. The contours of resultant displacement are shown in Figure 2.

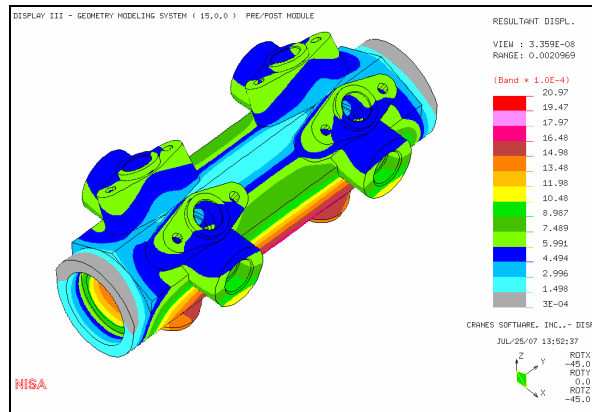


Figure 2 Resultant displacement contours

The problem considered for fluid flow analysis is that of a fluid mixing chamber. In this problem, fluid enters through two inlets into the chamber and exits through two outlets of same dimensions as that of the inlets. The solution methodology involves solving steady state, three dimensional momentum and continuity equations for pressure and velocity distributions. The number of hexahedral elements and degrees of freedom are 1,00,100 and 3,00,300 respectively. Figure 3 and Figure 4 give the velocity and pressure contours respectively. The results are shown for the front midsection and two side midsections across the inlet and outlet openings

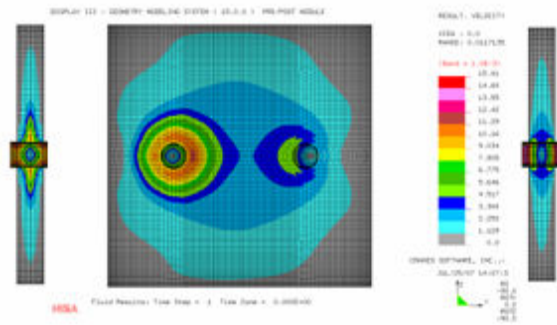


Figure 3 Resultant velocity contours

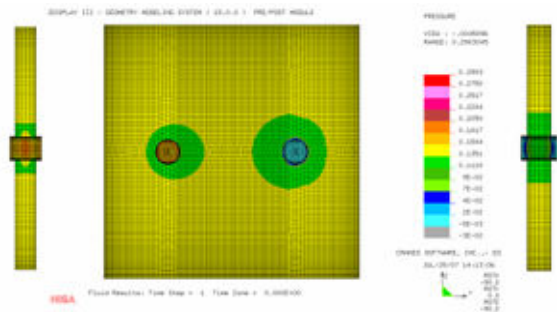


Figure 4 Pressure contours

Table 2 presents the comparison of solution time of PARDISO solver with that of existing built-in sparse solver. It also includes scalability of solver against the number of cores.

Table 2 Total Solution time in static analysis

DIRECT	PARDISO (MKL)			
	1	2	4	8
793.93	155.10	115	75.28	68.65

The solution time for solving linear equation is divided into three parts: 1) Reordering and symbolic factorization. 2) Numerical factorization. 3) Forward and backward substitution. This break-up in execution time of PARDISO solver is given in Table 3 and Table 4 for structural and fluid analyses, respectively.

Table 3 Break-up of solver execution time for Structural analysis

Remark	Threads			
	1	2	4	8
Reordering and symbolic Factorization	9.22	8.84	8.95	9.80
Factorization	118.58	80.31	40.70	33.97
Back Substitution	4.31	3.38	3.17	3.13

Table 4 Break-up of solver execution time for Fluid flow analysis

Remark	Threads			
	1	2	4	8
Reordering and symbolic Factorization	166.94	159.90	168.26	168.84
Factorization	1958.84	1118.54	682.40	627.21
Back Substitution	65.39	64.06	64.04	63.56

From Table 1 and Table 4, it can be observed that, the factorization time scales reasonably well with the number of cores. However, this is not the case with reordering and forward/backward substitution times.

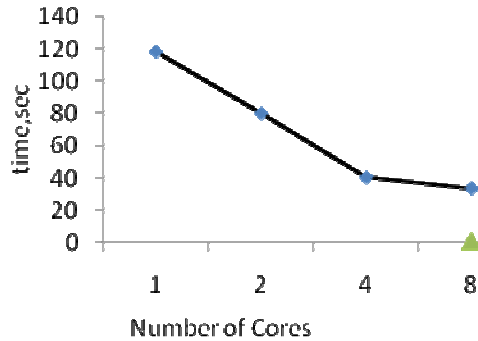


Figure 5 Factorization time versus number of cores

Figure 5 shows the factorization time versus the number of cores. From this figure, it is observed that the factorization time tends to flatten out after four cores.

5. CONCLUSIONS

In this work a solver plug-in frame work is proposed and implemented in NISA [11] for PARDISO solver in MKL.

The following observations can be made from the above study with respect to the PARDISO solver.

1) The PARADISO DIRECT solver is reasonably fast and the factorization time scales well up to four cores. However the reordering and the back substitution times appear to be same across cores.

2) Since PARADISO uses in-core Memory, its memory requirement increases with the size of the problem. This would be a limitation for solving large problems on desktop system with moderate memory (4 GB). This is

more so in case of fluid problems involving unsymmetrical matrices.

3) For large problems, the reordering time can be significant. Since reordering time does not scale up with the number of cores the overall performance may not improve appreciably with cores.

6. ACKNOWLEDGMENT

We like to thank Intel Corporation for providing the required computational facilities through Intel® Remote Access Service.

REFERENCES

1. Bathe K. J., 'Finite Element Procedures', Prentice hall of india New Delhi 1997.
2. Equation Solver Interface. <http://z.ca.sandia.gov/esi/>, 2005.
3. Tops Solver Component. <http://www-unix.mcs.anl.gov/scidac-tops/solver-components/tops.html>, 2006.
4. Fang (Cherry) Liu and Randall Bramley 'CCA-LISI: On Designing A CCA Parallel Sparse Linear Solver Interface', ACM/IEEE 21th International Parallel and Distributed Processing Symposium (IPDPS), Long Beach California, March 2007
5. Intel® Math Kernel Library (MKL), Reference Manual, 2007. <http://www.intel.com/software/products/mkl/>
6. Anshul gupta, 'WSMP: Watson Sparse Matrix Package Part I - Direct solution of symmetric sparse system', Tech Rep 21886 (98462) IBM T. J. Watson Research Center, Yorktown Heights, NY 2000.
7. Anshul gupta, 'WSMP: Watson Sparse Matrix Package Part II - Direct solution of symmetric sparse system', Tech Rep 21888 (98472) IBM T. J. Watson Research Center, Yorktown Heights, NY 2000.
8. P. R. Amestoy, I. S. Duff and J.-Y. L'Excellent (1998), 'Multifrontal parallel distributed symmetric and unsymmetric solvers', in Comput. Methods in Appl. Mech. Eng., 184, 501-520 (2000).
9. X. Li and J. Demmel. SuperLU DIST: 'A scalable distributed-memory sparse direct solver for unsymmetric linear systems'. ACM Transactions on Mathematical Software, 29:110140, 2003.
10. F.S. Smailbegovic, G. N. Gaydadjiev, S. Vassiliadis, 'Sparse Matrix Storage Format', Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc 2005, pp. 445-448, Veldhoven, the Netherlands, November 2005
11. NISA user's Manual, version 15.0, Cranes Software, Inc., Troy, Michigan, USA, 2007.

Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the “Intel® Compiler User and Reference Guides” under “Compiler Options.” Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101