Parallelization of a commercial Boundary/Finite Element Code using compiler directives

A. J. Svobodnik, Z.-S. Chen, E. Schnabler

Numerical Analysis and Design GmbH & Co KG, Landskrongasse 5, A-1010 Wien, Austria
email: {as,ch,es}@*NAD*work.at
web: http://www.*NAD*work.at
phone: +43.1.533.53.06-0

Abstract

Today in many engineering applications one of the key issues is computing time. Especially in computational acoustics many typical applications are very time consuming. One way to address this problem is to use more than one processor for execution. Thus it is obvious to try to parallelize an existing serial code to drastically reduce computing time. This paper will show the strategy to parallelize an existing analysis code using compiler directives as implemented in the HP C/ANSI Compiler for HP-UX11 for executing in 32-bit as well as in 64-bit mode. Furthermore we will show the excellent scalability and load balancing of the code based on some typical industrial applications. However, we will also focus on the problems we encountered during code development.

1 Introduction

There is a long tradition in computational acoustics to solve practical problems via the use of boundary element codes. The main advantages of this method are

simple meshing (due to two-dimensional meshes)
automatically fulfills the so-called “Sommerfeld-Condition” for exterior problems

Today there exist many more fields of applications which go far beyond the original research in former times. In the last two decades, there has been much research and progress to solve for coupled problems with respect to fluid structure interaction. Beside the classical boundary element method there is also a wide use of finite element methods, especially in acoustics for interior problems, infinite elements for exterior problems and other approximation methods.

Since 1994 we are working on the (further) development of a general purpose analysis code for computational acoustics entitled *NAD*work/Acoustics [i], including comprehensive pre- and postprocessing facilities and various interfaces to CAD- and CAE-systems. The primary goal of this code was the solution of uncoupled and coupled acoustic phenomenas using the boundary and finite element method, especially in the area of audio engineering and mechanical and electrical engineering. The software was first released in 1997. Details on the theory behind *NAD*work/Acoustics can be found in [ii] and [iii].

At this time we started a research project entitled *NAD*work/HPC with the primary goal to adapt our existing software products for the use in a high performance computing environment, especially for multiprocessor computers. This effort was mainly driven by two fields of applications of our software. One is the simulation of nonlinear effects like nonlinear stability analyses with large deflections and rotations, and the other is the calculation of sound radiation of vibrating structures. In the first one, the long computation times are due to the highly nonlinearities like contact and plasticity at finite strains. In the latter one, one has to solve for a coupled fluid-structure problem in the frequency domain for large problem sizes.

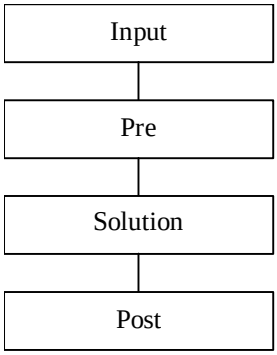
In a first step of this class of application the structural eigenfrequencies are calculated in the frequency domain of interest. In typical applications there arise systems with several 100k of unknowns and several hundreds (up to 2,000) eigenvalues. In further steps the motion of the body due to some excitement is calculated, and finally the propagation of sound waves in the surrounding fluid due to the motion of the structure are calculated. For the fluid (sound propagation) we get systems of symmetric, complex and fully populated (BEM) or sparse matrices (FEM). In the case of BEM we typically get up to 20,000 of unknowns, and in the case of FEM we get up to 1,000,000 of unknowns. Thus the term large scale analysis seems to be valid.

As already outlined in a previous paper [iv] in a first phase of the research project, our focus was on the solution phase of linear systems. Our target was to implement a software interface for our existing code to various solver libraries. On the one hand the effort was to make an interface to proprietary solvers from hardware vendors, like HP's MLIB, which usually use direct solvers, whereas on the other hand we started the development of iterative solver methods. This paper will now show the strategy for the parallelization of our existing code, and to show the increase in performance for some typical industrial applications. This paper will focus on aspects of parallelization of code segments relevant to acoustic calculations.

All work was done on a HP V2200, running under HP-UX 11.0, equipped with 8 processors, 2GB of memory and several GB's of harddisk space on an AutoRAID system.

2 The SW-Strategy

Typically, a computational acoustics code based on BEM or FEM consists of the following program phases:



During the *Input* phase the data describing the problem (nodes, elements, materials, loads, etc.) are being read. In the *Pre* phase the data structure of the global system matrix is being evaluated, which defines the storage scheme of the system of equations. Furthermore, the local element matrices of each element are calculated and assembled into the global system matrix. In the *Solution* phase the system of equations is being solved for one or more right-hand-side vectors. And finally in the *Post* phase the ultimate results like sound pressures or velocities are being carried out.

The main goals in the development of *NAD*work/HPC were beside reaching a good performance ease of programming (to reduce development time) and portability. So we decided to use a programming model based on a shared memory paradigm using compiler directives as implemented in the HP C/ANSI compiler via the use of `#pragma _CNX` directives. The disadvantage of missing portability is not a problem, as all `#pragma _CNX` directives we used have a pendant in `OpenMP [v]`.

The parallelization itself was a straight-forward task. In the phases *Pre* and *Post* there is a global element loop over all elements in the model which can simply be parallelized by using a `#pragma _CNX loop_parallel` directive. For some loops we needed a `#pragma _CNX no_side_effects` and/or a `#pragma _CNX critical_section` directive (to avoid synchronization problems when to different local elements contribute to the same global equation). Using the same methods we did a parallelization of the equation solver.

However, there where two major drawbacks on development time. The first was compile time. At the beginning of the project a compile time of up to four hours for one module compared to five minutes for the sequential compilation was a problem. We could solve this problem by restructuring the code.

The second problem was efficiency of the parallel code. A good efficiency, as we have now, could be reached via:

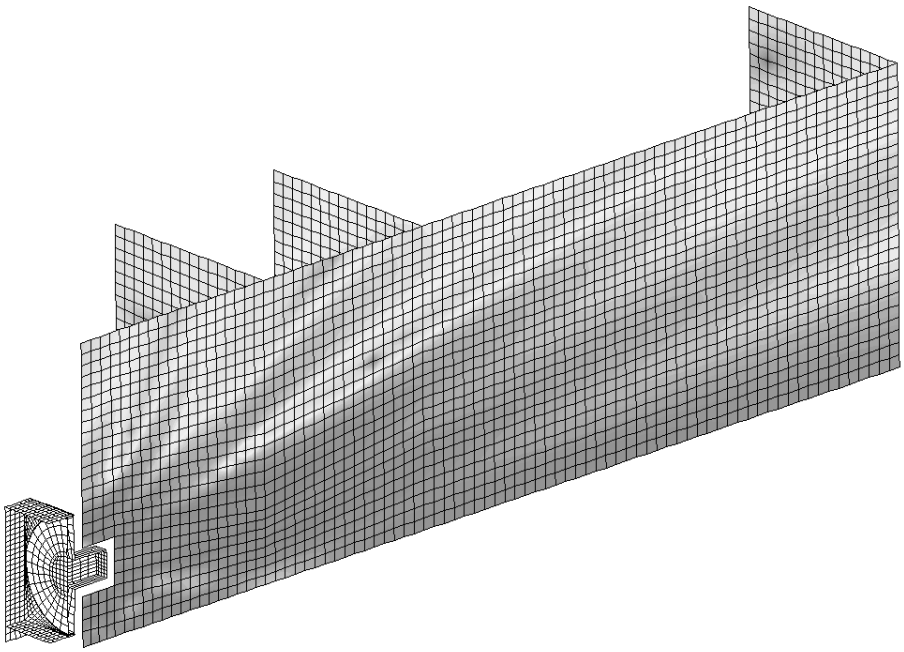
`#pragma _CNX loop_private` directives
code restructuring
`+Onodynsel` compile time option

The third problem was the unavailability of a debugger. HP debuggers cannot debug code compiled at optimization level `+O3` whereas `+O3` is needed for the parallel directives!

However, if we see now the obtained performance of our software, it was the right decision to use HP's `#pragma _CNX` directives.

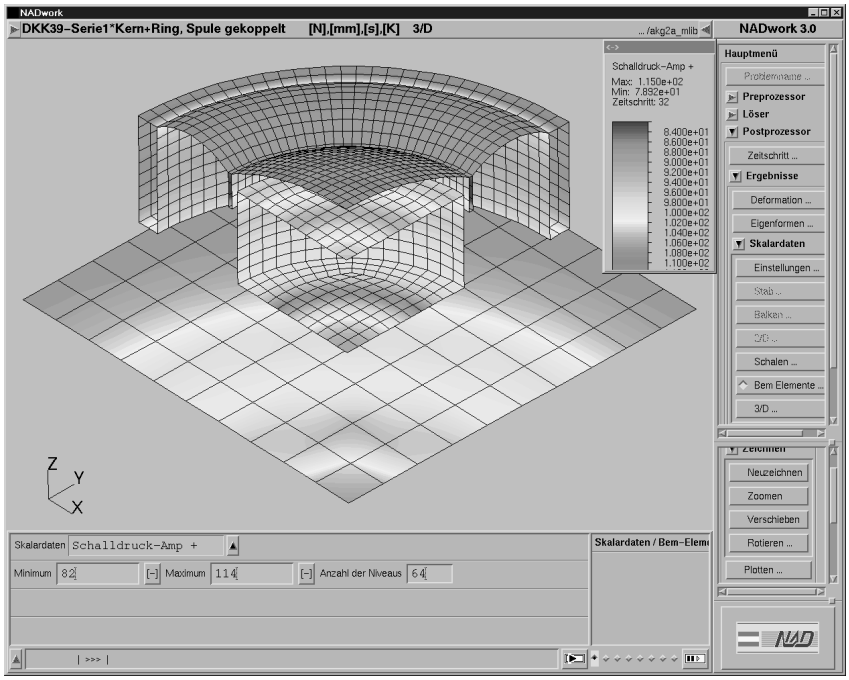
3 Benchmarks

In the following we will show some results we evaluated using the above described SW-strategy. We will show four real-life industrial examples for acoustic analyses. The first benchmark is an analysis of a typical mechanical engineering construction called *asa*, which describes the radiating sound of an oil cooling system driven by a fan. The boundary element model is shown in the following picture:



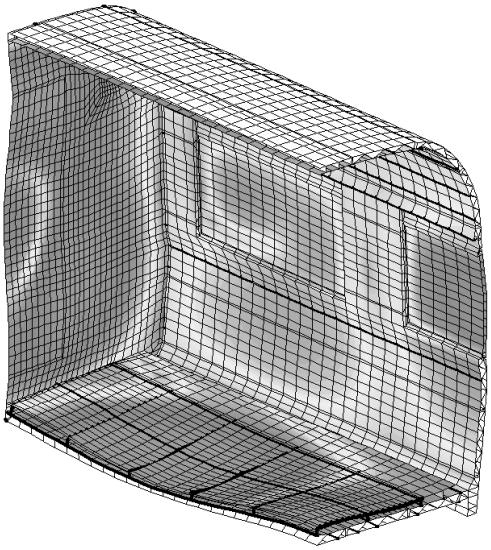
Results were carried out using a pure acoustic analysis, i.e. no coupling is considered. The model consists of 4,528 acoustic elements (1,156 boundary elements and 3,372 elements for post processing) resulting in an equation system of 1,406 of unknowns.

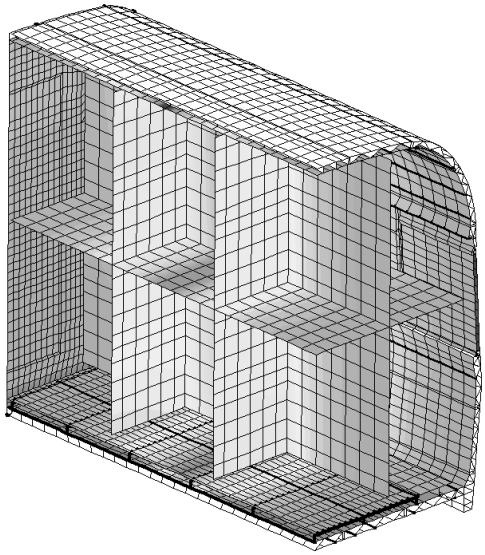
The second benchmark is a typical audio engineering application entitled *akg*, describing a transducer assembly as used in microphones `[vi]`. The boundary element model is shown in the following picture (only one quarter is shown):



Results were carried out using a two-sided coupled elastoacoustic analysis. The model consists of 1,104 structural elements (3/D-elements and shell elements) and 2,372 acoustic elements (2,326 boundary elements and 46 elements for post processing) resulting in an equation system of 5,622 unknowns for the structural part and 3,842 unknowns for the acoustic part.

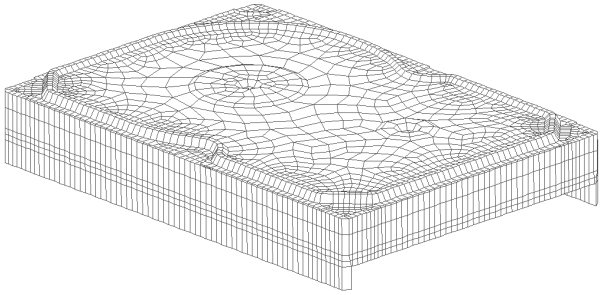
The third benchmark is a model of a railway chassis[1], entitled *DUEWAG*. The mixed boundary/finite element model is shown in the following pictures:

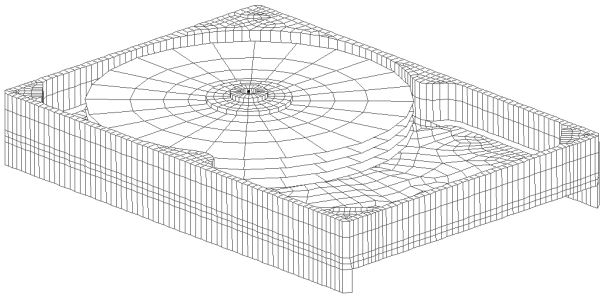




Results were carried out using a two-sided coupled elastoacoustic analysis. The model consists of 10,430 structural elements (beams, shells and solids) and 6,621 acoustic elements (4,809 boundary elements and 1,812 elements for post processing) resulting in an equation system of 45,630 unknowns for the structural part and 6,525 unknowns for the acoustic part.

The fourth benchmark is a typical electrical engineering application entitled *pmdm*, describing an analysis of a hard disk drive as widely used in PC's, workstations and servers. The mixed boundary/finite element model is shown in the following pictures:





Results were carried out using a two-sided coupled elastoacoustic analysis. The model consists of 10,386 structural elements (beams, shells and solids) and 6,911 acoustic elements (6,111 boundary elements and 800 elements for post processing) resulting in an equation system of 47,139 unknowns for the structural part and 10,229 unknowns for the acoustic part.

[1] Please note that this chassis represents only a segment of a much larger chassis. This was done for testing purposes only. However, this segment was also manufactured for comparing calculated and measured results.

The following table shows the evaluated results for all four benchmarks:

Benchmark	**asa**						
No. Procs.	Time (WALL)	Time (CPU)	CPU/WALL	Speed-Up I	Speed-Up II	Eff. I	Eff. II
1 (sequ.)	931,00	928,97	1,00	*1,00*	1,03	*100,00%*	103,11%
1 (par.)	960,00	958,89	1,00	0,97	*1,00*	96,98%	*100,00%*
2	512,00	1015,69	1,98	1,82	1,88	90,92%	93,75%
4	267,00	1044,92	3,91	3,49	3,60	87,17%	89,89%
6	188,00	1085,70	5,78	4,95	5,11	82,54%	85,11%
8	158,00	1210,82	7,66	5,89	6,08	73,66%	75,95%

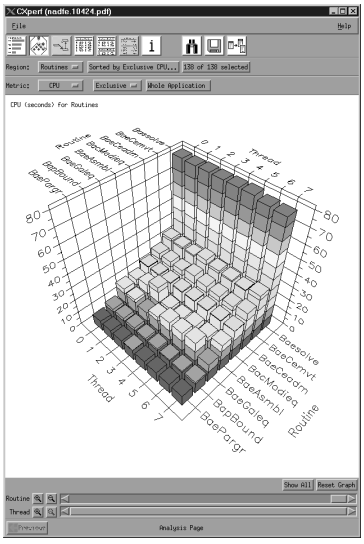
Benchmark	**akg**						
No. Of Procs.	Time (WALL)	Time (CPU)	CPU/WALL	Speed-Up I	Speed-Up II	Eff. I	Eff. II
1 (sequ.)	1336,00	1333,95	1,00	*1,00*	0,98	*100,00%*	98,05%
1 (par.)	1310,00	1308,03	1,00	1,02	*1,00*	101,98%	*100,00%*
2	693,00	1354,27	1,95	1,93	1,89	96,39%	94,52%
4	379,00	1426,48	3,76	3,53	3,46	88,13%	86,41%
6	287,00	1572,74	5,48	4,66	4,56	77,58%	76,07%
8	253,00	1792,89	7,09	5,28	5,18	66,01%	64,72%

Benchmark	**DUEWAG**						
No. Of Procs.	Time (WALL)	Time (CPU)	CPU/WALL	Speed-Up I	Speed-Up II	Eff. I	Eff. II
1 (sequ.)	14469,00	14403,21	1,00	*1,00*	1,06	*100,00%*	105,94%
1 (par.)	15329,00	15279,18	1,00	0,94	*1,00*	94,39%	*100,00%*
2	7337,00	14551,22	1,98	1,97	2,09	98,60%	104,46%
4	3951,00	15427,35	3,90	3,66	3,88	91,55%	96,99%
6	2841,00	16468,61	5,80	5,09	5,40	84,88%	89,93%
8	2343,00	17676,30	7,54	6,18	6,54	77,19%	81,78%

Benchmark	**pmdm**						
No. Of Procs.	Time (WALL)	Time (CPU)	CPU/WALL	Speed-Up I	Speed-Up II	Eff. I	Eff. II
1 (sequ.)	16682,61	16699,00	1,00	*1,00*	0,99	*100,00%*	99,47%
1 (par.)	16595,00	16582,84	1,00	1,01	*1,00*	100,53%	*100,00%*
2	8674,00	17120,12	1,97	1,92	1,91	96,16%	95,66%
4	4684,00	18084,88	3,86	3,56	3,54	89,04%	88,57%
6	3595,00	20489,50	5,70	4,64	4,62	77,34%	76,94%
8	3049,00	22762,37	7,47	5,47	5,44	68,39%	68,03%

All timing values represent total execution time including all program phases. Column *Speed-Up I* uses the sequential code for reference, whereas *Speed-Up II* uses the parallel code running on one processor for reference. Columns *Eff. I(II)* is simply the relation of the speed-up to the number of processors used. As can be seen the efficiency decreases while using more processors, as to be expected. However, we also see an efficiency up to 80% when using eight processors. Currently further benchmarking is going on to evaluate results using more than eight processors.

For all projects up to now where *NAD*work/HPC was used, we see a very good load balancing. In the following picture you can see a parallel performance analysis via the use of HP's `Cxperf`:



4 Conclusion

We have shown the SW-strategy to parallelize an engineering analysis code as used in computational acoustics. For some typical engineering applications benchmarks were presented. We could show the excellent performance behavior of the parallelized code with respect to scaling with number of processors and load balancing. It turned out that it was the right decision using HP's compiler directives for parallelization, though there where some drawbacks during code development due to some compiler deficiencies.

References

[i] Numerical Analysis and Design GmbH & Co KG, *NAD*work/Acoustics V3.0 Release Notes, 1999

[ii] Chen, Z.-S., Hofstetter, G., Mang, H. A.: A symmetric Galerkin formulation of the Boundary Element Method for acoustic radiation and scattering, Journal of Computational Acoustics, **25**, 219-241, 1997

[iii] Chen, Z.-S., Hofstetter, G., Mang, H. A.: A Galerkin-type BE-FE formulation for elastoacoustic coupling, Computer methods in applied mechanics and engineering, **152**, 147-155, 1998

[iv] Svobodnik, A. J., Schnabler E.: Interfacing a commercial Finite Element Code to HP's MLIB running on a V-Class System, HiPer '98, 1998.

[v] `http://www.openmp.org`

[vi] AKG Acoustics GmbH, Vienna, 1999