100

Dynamic Scheduling of Substructure Computations in an Industrial Production Environment

Petter E. Bjørstad, Jon Brækhus, and Jeremy Cook

1 Introduction

Domain decomposition of large structural analysis problems based on physical substructures modeled separately can be used to get a good ordering for the direct Cholesky factorization of the global stiffness matrix [Prz85], [SBG96]. For large substructure elimination trees one can further increase the computational speed by processing independent substructures in parallel [BBH91]. However, this technique often results in substructures of very different size leading to an important load balancing problem. This paper discusses experience in an industrial setting using a workstation cluster.

The SESAM[1] software package is a general, structural analysis finite element package for static and dynamic linear analysis. The linear equations are solved by a module SESTRA using a multilevel superelement technique [Det94]. SESAM has users worldwide with focus on the analysis of ships and offshore structures. A failure in such structures can have very serious consequences. Thus, the demand for more accurate analysis is a driving force towards efficient and more robust computer implementations.

The stiffness matrix of a superelement (which here always corresponds to a substructure of the full structure) takes the form

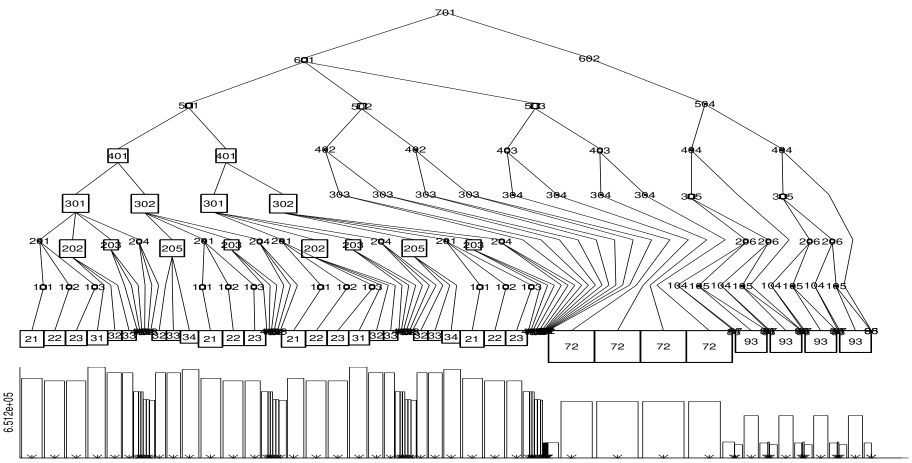
$$\begin{pmatrix} K_{ii} & K_{ir} \\ K_{ir}^T & K_{rr} \end{pmatrix}. \tag{1.1}$$

Here i denotes an internal node while r is a node on the substructure boundary (connecting it to other substructures). The computation lends itself naturally to a two-level parallel implementation, a coarse grain level where independent substructures can be computed in parallel and a fine grain level where the linear algebra within a substructure calculation is parallelized. This calculation consists of forming the Schur

1 SESAM is developed and marketed by Det Norske Veritas a/s.

Ninth International Conference on Domain Decomposition Methods
Editor Petter E. Bjørstad, Magne S. Espedal and David E. Keyes

Figure 1 The complete substructure tree of the Troll model. The tree has 147 tasks and must be used in the dynamic back substitution algorithm. There are many identical tasks and only 48 are needed in the scheduling of the matrix factorization. The critical paths are shown in the bar graph below the tree. The size of the boxes around each superelement indicates the computational load.



complement

$$S = K_{rr} - K_{ir}^T K_{ii}^{-1} K_{ir}. \tag{1.2}$$

In practice, this is carried out by a Cholesky factorization of the interior part K_{ii}, a forward triangular solve with K_{ir} as the right hand side, and a matrix multiplication where the resulting matrix K_{rr} is symmetric. We overwrite K_{ir} in this process and note that we only compute the Schur complement for substructures that are different. The same substructure can be used as a building block in several places when forming the global model with a considerable computational savings, see Figure 1.

The overall parallel strategy was outlined in [Bjø87] and the coarse grain part was implemented and turned into a commercial software product by Hvidsten [Hvi90]. A fine grain parallel implementation is described in [CBB96].

In this work we are only interested in testing the coarse grain parallel implementation in an industrial setting in order to report on the actual benefit of using a cluster of workstations. We will further report on the most important factors that must be taken into account in order to get satisfactory parallel performance. We have chosen a finite element model of the Troll[2] offshore gas platform as our test case. The model of Troll that we will use is of medium size having 701508 degrees of freedom representing the 369 meter tall concrete base of the platform.

2 Troll went into production in 1995, with a height of 472 meters and a total weight of more than 10^9 kg it is the largest structure made by man and subsequently moved to its final destination. See http://www.shell.no/troll/ for pictures and more information.

2 Scheduling of Parallel Tasks

We use a dynamic scheduling algorithm as opposed to an a priori statically determined scheduling. Each time a processor is available and requests a new task, SESTRA will execute its scheduling algorithm. Two factors must be carefully examined by this algorithm:

- The computational size of a task
- The computational power of a workstation.

The size of a task is available as an estimate of the required number of floating point operations to form the Schur complement given in (1.2). This estimate is derived from the number of internal and boundary degrees of freedom where also the bandwidth of K_{ii} and empirical factors to compensate for fill during the factorization are included. Ideally, this estimate could be even more precise if based on a symbolic factorization phase, but this would require accessing the data twice which may carry a significant cost. Also, the amount of disk space needed to store the superelement matrices in factored form must be estimated. Thus, the computational task is estimated with respect to both time and space.

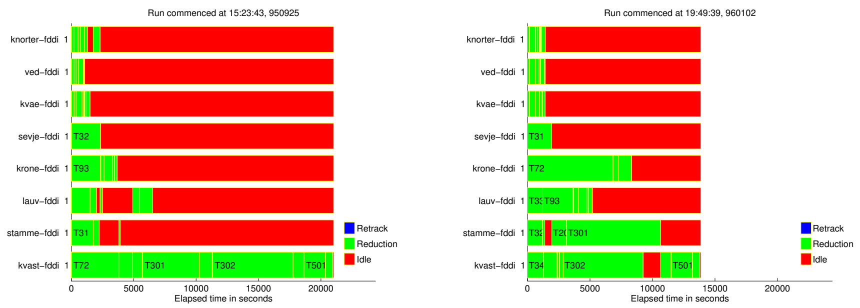
The power of a workstation closely matches the task estimate. The algorithm is given a list of 'realistic' floating point speeds of each workstation. Since this speed unlike the 'peak speed' often quoted by vendors, depends on actual workstation load it may even be updated dynamically by comparing actual elapsed time with the predicted one after each task completion. The space variable of a workstation corresponds to the available local disk space and the size of its memory. If the substructure cannot be stored locally, then this can greatly affect I/O speed and thereby overall elapsed time for a given calculation. Similarly, a large memory will be used to cache disk I/O on all modern workstations. Thus, this factor can have a significant impact on the computing time as well.

In our first algorithm, the largest available task was scheduled first. It turned out that this heuristic was too primitive and our current algorithm uses the estimated critical path with respect to computational loads as the basis for task scheduling. That is, the possible tasks will always be considered starting with the substructure which is at the end of the longest critical path to the root of the tree. The algorithm will also keep a table holding the estimated time until each workstation has finished its current task. Informally stated, the scheduling proceeds as follows:

MY workstation is idle and requests a new task:

1. Select the task at the end of the current critical path
2. Estimate the completion time for this task on MY workstation
3. Estimate the completion time for all other (busy) workstations on this task, taking into account the estimate for completion of their current work
4. If I complete first, assign this task to ME and exit here, otherwise update the completion estimate of whoever will be faster than ME with this task
5. If there are more available tasks go to 1
6. If no task found for ME, go idle and check back later.

Figure 2 Scheduling of superelement tasks on 8 nodes. To the left, the time history from September 95, with the largest task first and too much priority to the fastest processor. To the right, the time history from January 96 with priority to tasks on the critical path.



3 Experience with the Scheduling

In this section we discuss our test problem using the dynamic scheduling of substructures as outlined above.

Our cluster of workstations consists of eight DEC Alpha EV45 with 233 MHz processors. One machine, kvast, has 512 MB memory while the other seven have 128 MB. Each machine has a separate FDDI segment all of which are interconnected by a Gigaswitch. Four machines each have 4 GB of local disk while the other four have only one gigabyte of local disk that can be used in the computation. On the single 512 MB workstation the analysis of Troll takes about 9 hours while the time increases to 14 hours on a 128 MB machine with 4 GB of local disk. Thus, SESTRA runs almost 60 percent faster on the machine with large memory due to efficient caching of file I/O to the free memory. The cluster is therefore inhomogeneous with respect to our scheduling algorithm despite the fact that each processor has the same theoretical speed. A substructure task will always use a disk that is large enough to hold its local files. Our algorithm from the previous section will rate the 512 MB machine as more powerful and dynamically further adjust the power of the machines depending on whether they need to access a local or remote disk for their current task.

The substructure tasks from Troll are of different sizes. The smallest task takes 9 seconds to complete, whereas the largest needs about two hours on the fastest machine. That is, we have about a factor 1000 in difference between the largest and the smallest tasks in this model.

The improvement from always attacking the largest available task to using a set of updated critical paths as a basis for scheduling can be seen in Figure 2. The left picture shows a situation where the largest task is scheduled first and the most powerful workstation ends up with too much work. We observe that all the tasks on the path after 72 are quite light, while tasks 301 and 302 higher up in the tree are quite heavy. The slower machines do not participate as much as they could and the total analysis time is 5 hours and 50 minutes. The right part of the figure shows a much better distribution resulting from our algorithm given earlier. The total time has been reduced by 34 percent to 3 hours and 51 minutes.

Table 1 Elapsed time for the factorization and back substitution phase.

Workstations	Matrix Factorization	Back Substitution
1	8h 57m	2h 55m
2	6h 05m	-
4	4h 03m	1h 32m
8	3h 51m	0h 54m

Table 1 shows the elapsed time with our scheduling algorithm for different number of participating workstations. The gain from using more than 4 workstations is only slight, and is clearly not cost effective. There are only a limited number of branches with a significant amount of computation so adding more processors does not help unless we also exploit the finer grain parallel work when computing (1.2) inside each substructure task. However, for many customers having 2 to 8 workstations available, this improvement means that they can carry out two analysis each day instead of one overnight run.

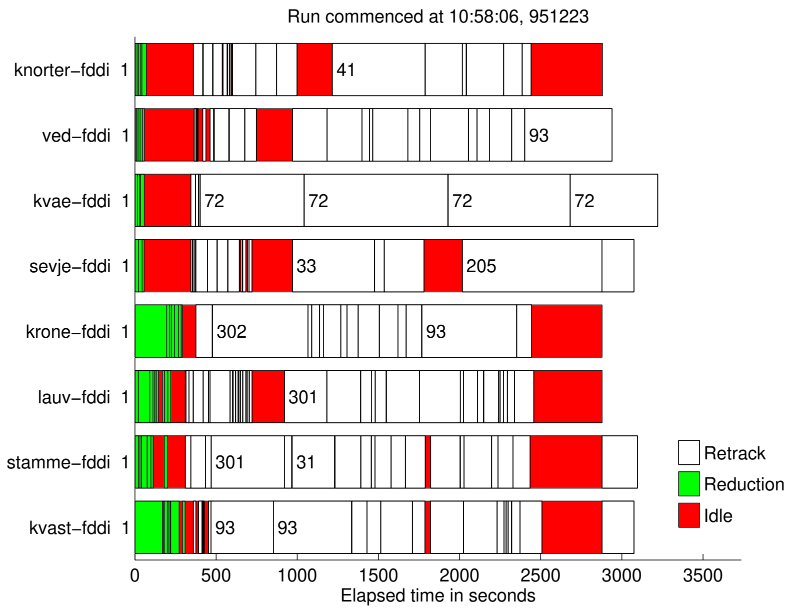
Often current jobs are larger than Troll, requiring more than 25 hours on a single workstation. This may be reduced to less than 10 with a 4 workstation cluster.

4 Parallel Back Substitution and New Loads

The main purpose of structural analysis calculations is to study how a given structure reacts to different loading conditions such as wind, wave, surface or simply point loads. Often, the main interest is focused on the response of specific substructures only. After the factorization phase has been completed such questions can be studied with considerable flexibility. Given a new right hand side (load condition), the analysis will first again proceed up the tree in Figure 1 starting at the substructure(s) subject to the load in order to transform the right hand side consistently with the elimination of interior degrees of freedom when we transformed the original system from equation (1.1) to (1.2) during the factorization step. This calculation involves a triangular solve and a matrix vector multiplication within each task along the path. In engineering terms this is called load reduction. Next, we must transverse the tree back, starting from the root and again do a back substitution and a matrix vector multiplication for each substructure along the path to the substructure whose response we are interested in. This process differs from the factorization step by:

- Repeated occurrences of substructures are now separate tasks. The number of tasks in our test model increases from 48 to 147.
- When traversing the tree from the root to the leaves, we get more and more parallel tasks as we go.

Figure 3 Task scheduling for retracking



- The size of each task depends more weakly on the number of equations. It is therefore easier to achieve good load balancing.
- All tasks grow linearly with the number of load cases.
- Repeated superelements cannot access the same substructure data simultaneously. Dynamic scheduling should therefore give priority to repeated tasks.
- The algorithm used in Figure 3 just searches a list of available tasks and grabs the first available, but more advanced logic have also been implemented.

We report on a case with complete back substitution for the entire structure with 20 different load conditions. The computing time is given in Table 1 and shows good utilization of all eight workstations. We note that this part of the computation is about 30 percent of the factorization phase, thus this time can be even more significant in an analysis that may have several hundred different loading conditions. From Figure 3 we see that even a very simple dynamic scheduling algorithm works well in this case.

5 Conclusions

We have documented the benefits of dynamic substructure scheduling within a large commercial structural analysis code. Our scheduling achieves near optimal results for the example tested in the factorization phase. This approach automatically incorporates knowledge about the actual load present in the workstation environment. The back substitution phase has excellent parallel properties when there are many right hand sides and a substantial part of the complete solution is wanted.

We have shown that a large industrial application, traditionally run on large supercomputers, can be used successfully for *industrial strength* problems in a workstation environment giving a very cost effective solution. Using a workstation environment also carries the added benefit that large result files can be postprocessed on the same architecture.

Acknowledgement

The authors express thank to A/S Norske Shell, the development operator of Troll GAS for permission to use the Troll GBS finite element model. The model was made by Norwegian Contractors before building the platform's gravity base structure. This work was carried out with funding from the EC-ESPRIT III EUROPORT subproject 8586.

REFERENCES

[BBH91] Bjørstad P. E., Brækhus J., and Hvidsten A. (1991) Parallel substructuring algorithms in structural analysis, direct and iterative methods. In Glowinski R., Kuznetsov Y. A., Meurant G. A., Périaux J., and Widlund O. B. (eds) *Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 321–340. SIAM, Philadelphia, PA. Proceedings from the fourth international symposium on domain decomposition, Moscow May 1990.

[Bjø87] Bjørstad P. E. (1987) A large scale, sparse, secondary storage, direct linear equation solver for structural analysis and its implementation on vector and parallel architectures. *J. Parallel Comp.* (5).

[CBB96] Cook J., Bjørstad P. E., and Brækhus J. (April 1996) Multilevel parallel solution of large, sparse finite element equations from structural analysis. In Liddel H., Colbrook A., Hertzberger B., and Sloot P. (eds) *High-Performance Computing and Networking*, volume 1067, pages 404—412. Springer. Lecture Notes in Computer Science.

[Det94] Det Norske Veritas – Sesam, P.O. box 300, N-1322 Høvik, NORWAY (March 1994) *SESAM technical description*.

[Hvi90] Hvidsten A. (1990) *On the Parallelization of a Finite Element Structural Analysis Program.* PhD thesis, Computer Science Department, University of Bergen, Norway.

[Prz85] Przemieniecki J. S. (1985) *Theory of Matrix Structural Analysis.* Dover Publications, Inc., New York. Reprint of McGraw Hill, 1968.

[SBG96] Smith B. F., Bjørstad P. E., and Gropp W. (1996) *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press.