A parallel p-version FE-approach for structural engineering

Martin Rücker, Manfred Krafczyk, Ernst Rank

LS Bauinformatik, Department of Civil Engineering and Surveying
Technical University of Munich, Munich, Germany

Abstract

This paper describes a parallel p-version FEM approach for civil and structural engineering. After a brief introduction to the p-version of the finite element method we will consider a condensation technique for the interior degrees of freedom which decreases the size of the global equation system essentially. The implementation of the parallel approach described in the third part is concentrated on the more time consuming element based computation, the solution of the global equation systems is sequential. Finally, the efficiency of the implemented algorithm is shown in two numerical examples.

1 Introduction

The p-version of the finite element method has been investigated very intensively during the past 15 years and it has turned out to be superior to the classical h-version in a significant number of fields of practical importance. It has especially offered advantage when the finite element model is to be coupled to a structural model being defined in a CAD-environment. We will discuss results of the Esprit project INSIDE (Ref.No. 20216) [?], where a parallel p-version finite element code is implemented as one of several tasks in a civil engineering CAD-environment.

2 The p-version of FEM

While in the standard h-version of the finite element method the mesh is refined to achieve convergence, the polynomial degree of the shape functions remains unchanged. Usually low order approximation of degree p=1 or p=2 is chosen. The p-version leaves the mesh unchanged and increases the polynomial degree of the shape functions locally or globally. In most implementations a hierarchical set of shape functions is applied, providing a simple and consistent facility of implementation in 1-, 2- or 3-dimensional analysis. Oscillations in the approximate solution, which could be expected when working with high order shape functions can be avoided by using properly designed meshes. Guidelines to construct these meshes *a priori* can often be given much easier for the p-version than for the h-version [?]. For linear elliptic problems it was also proven mathematically that a sequence of meshes can be constructed so that the approximation error only depends on the polynomial degree p and not on the order of singularities in the exact solution.

Following [?], our p-version implementation uses hierarchical basis functions, which can easily be implemented up to any desired polynomial degree. Starting with the two linear shape functions on a standard element $[-1,+1]$, a whole *family* of shape functions in one space dimension is obtained by definition of a sequence of polynomials P_i with degree i satisfying the condition

$$\{P_i(\zeta) \mid i \geq 2\,,\, \zeta \in [-1,1], P_i(-1) = P_i(1) = 0\} \qquad (1)$$

A possible choice for P_i is given by integrals of the Legendre polynomials defined by

$$P_i(\zeta) \;=\; \int_{t=-1}^{\zeta} L_{i-1}(t)\,dt \qquad (2)$$

with

$$L_n(x) \;=\; \frac{1}{2^{n-1}(n-1)!}\,\frac{d^n(x^2-1)^n}{dx^n} \quad \text{for} \quad n \geq 1 \qquad (3)$$

A hierarchical basis has an immediate consequence on the structure of the resulting stiffness matrix. If equations are ordered so that all linear modes get numbers 1 to n_1, all quadratic modes get numbers n_1+1 to n_2 and so on, stiffness matrices corresponding to polynomial order 1 to $p-1$ are submatrices of the stiffness matrix corresponding to polynomial order p.

A construction of a hierarchical family of two-dimensional (and similarly three-dimensional) elements is quite straightforward. Standard shape functions are defined on a standard square $[-1,1]\times[-1,1]$ in local coordinates ζ, η by a tensor product calculation. They can be grouped into three classes. Elements in the first group are called *basic* modes being the 'usual' bilinear shape functions. The second class are *edge* modes. Hierarchical modes for edge $\eta = 1$ (and analogously for the other edges of the standard element) are defined by

$$H_i(\zeta,\eta) \;:=\; P_i(\zeta)(\eta+1)/2 \qquad (4)$$

Edge modes are identically zero on all edges except on their defining edge.

The third class are *bubble modes* being given by

$$H_{kl}(\zeta,\eta) \;=\; P_k(\zeta)\cdot P_l(\eta) \qquad k,l \geq 2 \qquad (5)$$

and disappearing on all edges of the element. Figure ?? shows a linear nodal mode, an edge mode of degree three and a bubble mode of degree five.

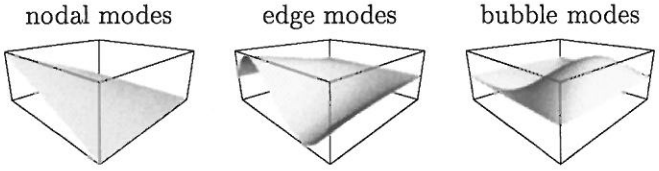

Figure 1: Shape functions for quadrilateral elements

Another main difference between h- and p-version finite-element methods lies in mapping requirements. Because in the p-version the element size is not reduced as the degrees of freedom are increased the description of the geometry must be independent of the number of elements. This results in the necessity to construct elements with an exact representation of the boundary. The isoparametric mapping, used in standard finite-element formulations, can be seen as a special case of mapping with the *blending function method* (see e.g. [?, ?]). Following these ideas element boundaries can be implemented as arbitrarily curved edges.

3 Parallelization of the p-version finite element analysis

Typically, the FEM computation can be divided into several subprocesses requiring different amounts of time:

- pre-processing : T_{pre}
- computation of element stiffness matrices : T_{comp}
- solution of the generated equation system : T_{solv}
- post-processing : T_{post}

In contrast to the h-version, in the p-version finite element method the computation of the element stiffness matrices and the post-processing usually takes much more time than solving the global equation system ($T_{comp} \gg T_{solv}$, $T_{post} \gg T_{solv}$). Thus, in the case of high polynomial degrees, the parallelization can be concentrated on the element computations in the presolution phase and on postprocessing and use a sequential algorithm for the remaining equation solution.

3.1 The primal subdomain implementation (PSI) for the p-version with high p-degrees

The implemented domain decomposition for high p-degrees (p $\geq$ 4) is a variant of the PSI-algorithm presented in [?]. To describe this algorithm, let us consider a *two-level domain decomposition.* On the first level, the finite element mesh Ω is decomposed into n non-overlapping subdomains ($\overline{\Omega}_j$, $j = 1, \ldots, n$), each consisting of a set of elements being identified with non-overlapping subdomains of level two (Ω_j, $j = 1, \ldots, s$). All elements of the first *level one* subdomain are numbered first and so on. Each p-element has nodal-, edge- and bubble-modes and a numbering of all degrees of freedom can be chosen so that all bubble modes for all elements get the lowest numbers, whereas all nodal- and edge modes are numbered last. Note that these modes correspond to 'interface nodes' in a domain decomposition for the h-version, whereas they do *not* necessarily correspond to interfaces of the *level one* subdomains $\overline{\Omega}_j$.

Now, the equation of equilibrium $Ku = f$ for the p-version finite element method can be written as

$$\begin{bmatrix} K_{ii}^{(1)} & 0 & & 0 & \widetilde{K}_{ib}^{(1)} \\ 0 & K_{ii}^{(2)} & & 0 & \widetilde{K}_{ib}^{(2)} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & K_{ii}^{(s)} & \widetilde{K}_{ib}^{(s)} \\ \widetilde{K}_{bi}^{(1)} & \widetilde{K}_{bi}^{(2)} & \cdots & \widetilde{K}_{bi}^{(s)} & K_{bb} \end{bmatrix} \begin{bmatrix} u_i^{(1)} \\ u_i^{(2)} \\ \vdots \\ u_i^{(s)} \\ u_b \end{bmatrix} = \begin{bmatrix} f_i^{(1)} \\ f_i^{(2)} \\ \vdots \\ f_i^{(s)} \\ f_b \end{bmatrix} \tag{6}$$

with

$$K_{bb} = \sum_{j=1}^{s} \widetilde{K}_{bb}^{(j)} \qquad \text{and} \qquad f_b = \sum_{j=1}^{s} \widetilde{f}_b^{(j)} \tag{?? a,b}$$

where

$$\widetilde{K}_{bb}^{(j)} = B_b^{(j)^t} K_{bb}^{(j)} B_b^{(j)}, \quad \widetilde{K}_{ib}^{(j)} = K_{ib}^{(j)} B_b^{(j)}, \quad \widetilde{K}_{bi}^{(j)} = B_b^{(j)^t} K_{bi}^{(j)}, \quad \widetilde{f}_b^{(j)} = B_b^{(j)^t} f_b^{(j)}. \tag{?? a-d}$$

$K_{ii}^{(j)}$ correspond to the bubble modes and $K_{bb}^{(j)}$ to the nodal and edge modes of Ω_j, while $K_{ib}^{(j)}$, $K_{bi}^{(j)}$ correspond to the interaction of these mode groups. $\widetilde{K}_{bb}^{(j)}$, $\widetilde{K}_{ib}^{(j)}$, $\widetilde{K}_{bi}^{(j)}$ and $\widetilde{f}_b^{(j)}$ are the element matrices and vectors, respectively, and $B_b^{(j)}$ is the connectivity matrix of element j mapping local nodal and edge mode numbers to their global numbering.

The concept behind this p-version PSI is exactly the same as that of the corresponding h-version implementation, i.e. to reduce the initial system of equations to a system comprising only the edge and nodal d.o.f.. Applying a static condensation to the subdomain internal d.o.f., Equation (??) can be transformed into the following problem for the nodal and edge unknowns:

$$\left(K_{bb} - \sum_{j=1}^{s} \widetilde{K}_{bi}^{(j)} K_{ii}^{(j)^{-1}} \widetilde{K}_{ib}^{(j)} \right) u_b = f_b - \sum_{j=1}^{s} \widetilde{K}_{bi}^{(j)} K_{ii}^{(j)^{-1}} f_i^{(j)} \tag{9}$$

or

$$S u_b = \widehat{f}_b \tag{10}$$

where

$$S = \sum_{j=1}^{s} \widetilde{S}^{(j)} = \sum_{j=1}^{s} B_b^{(j)^t} S^{(j)} B_b^{(j)} \tag{11}$$

$$S^{(j)} = K_{bb}^{(j)} - K_{bi}^{(j)} K_{ii}^{(j)^{-1}} K_{ib}^{(j)} \tag{12}$$

$$\widehat{f}_b = f_b - \sum_{j=1} K_{ii}^{(j)^{-1}} f_i^{(j)} \tag{13}$$

Matrix S is the Schur complement of K_{bb} in K and each subdomain matrix $S^{(j)}$ corresponds to a local static condensation operator or a local Schur complement.

Note now that all element Schur complements can be formed in parallel without any communication, so that for this reduction operation s processors (s is the number of elements) could be used. Yet, usually s is greater than the number of available processors n, so elements are grouped together to subdomains of *level one* and all Schur complements for elements in one of these subdomains computed sequentially by one processor.

As for high p-degrees bubble modes take the major part of all degrees of freedom, the size reduction of this element condensation is very significant. Furthermore, it was proven e.g. in [?], that this Schur complement is an excellent preconditioning for the remaining equation system, being solved sequentially in our implementation and taking only a minor part of the overall computational time. We use a PCG-solver for this final solution, either with incomplete Cholesky preconditioning or with simple diagonal scaling.

3.2 Implementation

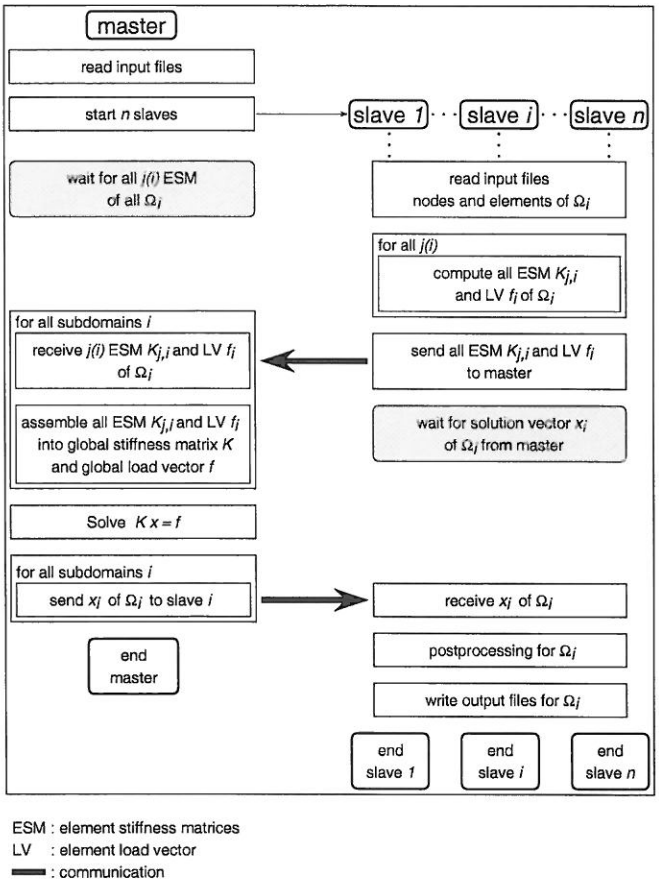

Figure 2: The parallel algorithm

The parallel algorithm is depicted in figure **??**. Our implementation is based on the message passing software PVM [?]. The two different process types, 'master' and 'slave' program, are unified into one code. After reading the global input files (geometry, domain decomposition, loading etc.) the master starts one slave process for each subdomain. The slaves select all data from the input files being relevant for their subdomain and start computation of the corresponding element matrices and load vectors. The slaves do their computation completely independantly without any communication, yielding a perfectly parallel computation in this phase. After elimination of the interior degrees of freedom each slave process sends its element matrices and load vectors to the master process, which assembles and solves the global equation system in the sequential part of the algorithm. The master process finishes after sending all element solution vectors of each subdomain to the corresponding slave processes. Just as the presolution phase, the postprocessing for each subdomain requires no communication between slaves or slaves and master. A final backward substitution for the interior degrees of freedom ends the computation for each slave.

4 Numerical examples

The efficiency tests are performed on a workstation cluster consisting of eight HP9000/C 100 (7×64 Mb RAM,

1×128 Mb RAM) connected by a standard 10 Mbit-Ethernet. The reference computations are made on the same workstation type with 256 Mb RAM. The computational times in the diagrams are given in real time. The absolute *speedup* $S(n)$ is defined by

$$S(n) = \frac{T(1)}{T(n)}$$

with

$T(1)$:computational time of the *sequential* program
$T(n)$:computational time of the *parallel* program using n processors

and the absolute *efficiency* $E(n)$ is defined by

$$E(n) = \frac{S(n)}{n}$$

(see [?]). Note that the efficiency for the implemented parallel algorithm is machine independent, because using a more powerful machine leeds to a speedup in the sequentiell part as well as in the parallel part. Due to the minimized communication and parallel overhead the difference in speed between the sequential and the parallel algorithm running on one single processor can be ignored.

In the following examples the variation of the number of subdomains/processes and the polynomial degree are studied. The domain decomposition was done using the public domain software METIS [?]. Note that the decreasing number of subdomains for high polynomial degrees is restricted by the memory of the 'slave' processors. The computational times for one subdomain are computed on a single processor. For two to seven subdomains each process runs on its own processor. In case of eight subdomains the master process and one slave process share one processor. Because of the fact that the time where both master and slave are in an active computing mode is very small, the influence on the computational speed is negligible.

The two examples are both plate problems, which are the main field of applications, but a comparable behaviour of the algorithm for plain stress/plain strain problems is expected. For all computations the full ansatz space is used.

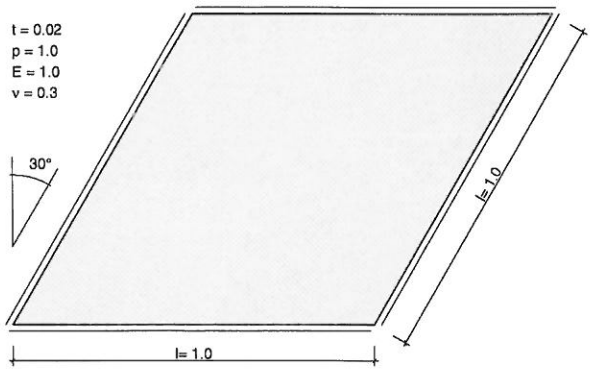

Figure 3: Rhombic plate – system

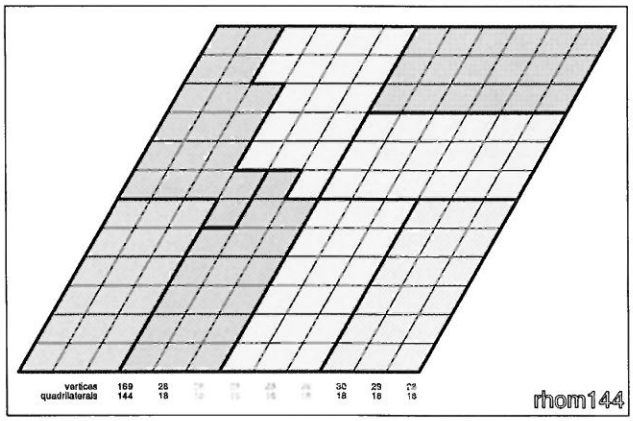

Figure 4: Rhombic plate – Finite element mesh and domain decomposition for 8 subdomains (by METIS [?])

Table ?? shows the number of degrees of freedom, the computational time, and the memory used for two different sequential program versions, the standard version and special version for large problems. The second 'low memory' version does not keep the reduced element stiffness matrices after assembly to the global stiffness matrix for post-processing. But computing the element matrices twice leads to an inacceptable computational time.

4.1 The rhombic plate

The system of this extensively studied benchmark test [?, ?] is depicted in figure ??. The finite element mesh consists of 144 regular quadrilateral elements. The mesh and an example for the domain decomposition are shown in figure ??. A more regular domain decomposition than the one shown in the example is possible but for the implemented algorithm of no importance.

p	dof	standard time [s]	standard memory [Mb]	low memory time [s]	low memory memory [Mb]
4	7011	25.41	10.26	31.10	7.11
5	10923	47.57	15.68	61.30	9.20
6	15699	83.87	23.63	114.07	11.68
7	21339	145.04	34.87	213.46	14.55
8	27843	252.96	50.30	382.05	17.82
9	35211	447.45	70.93	716.90	21.50
10	43443	759.71	97.87	1277.61	25.60
11	52539	1344.43	132.39	2346.70	30.12
12	62499	2147.05	175.85	3847.64	35.07
13	73323	3528.39	229.71	6289.21	40.47
14	85011	–	–	9748.10	46.32
15	97563	–	–	14891.34	52.64
16	110979	–	–	22015.68	59.44
17	125259	–	–	31339.54	66.73

Table 1: Rhombic plate – Degrees of freedom, time, and memory for various polynomial degrees – sequential computation

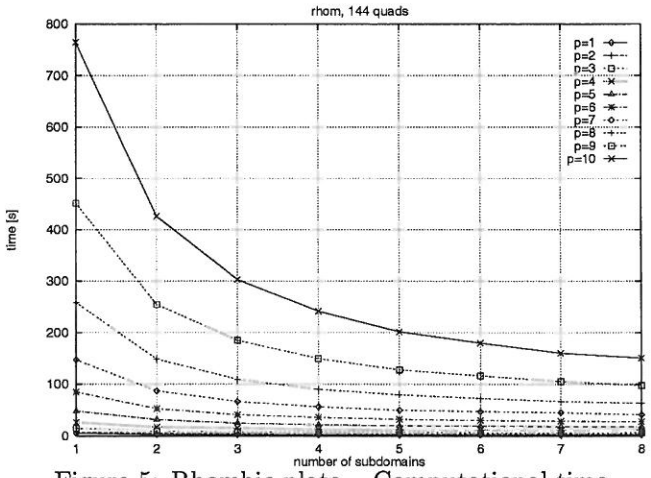

Figure 5: Rhombic plate – Computational time

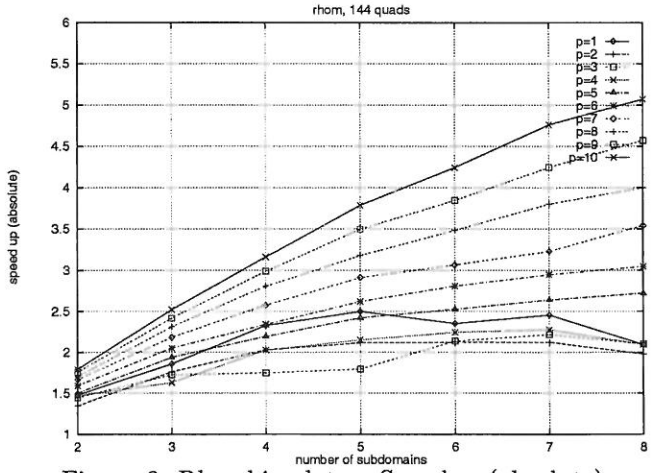

Figure 6: Rhombic plate – Speedup (absolute)

Regarding figure ?? it can be observed that the computational time for the parallel computation decreases with increasing number of subdomains for a moderate polynomial degree.

As expected the speedup increases with the polynomial degree. Because the computation on element level is perfectly parallelized the limit of computational time is basically specified by the sequential part.

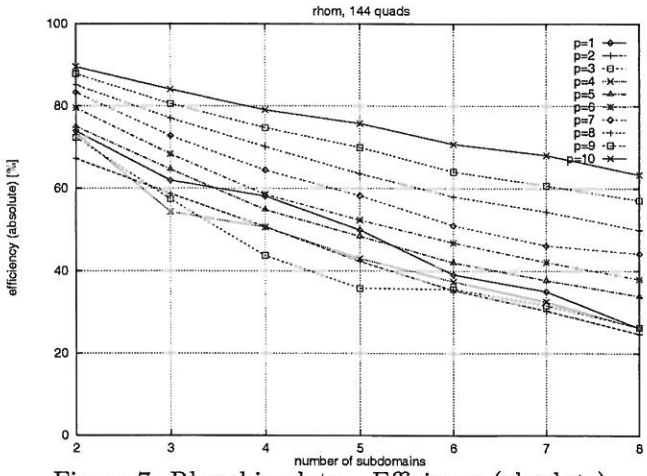

Figure 7: Rhombic plate – Efficiency (absolute)

4.2 A car park

The second example is the complex plate of a car park. The plate is supported by 30 columns modelled by mesh independant elastic foundations and several soft simple supported edges (see figure ??).

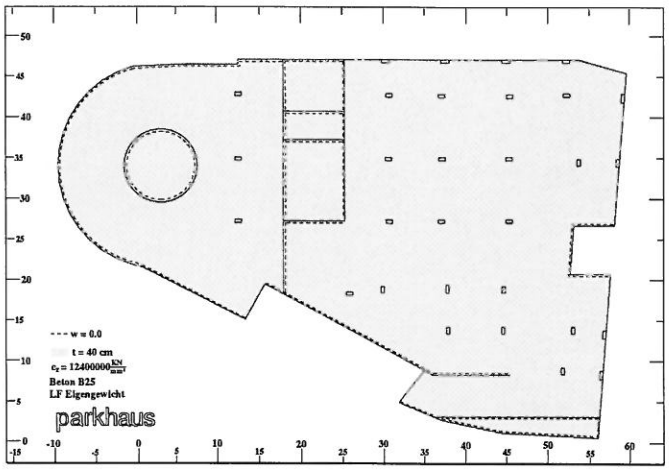

Figure 8: Car park – system

The finite element mesh with 104 quadrilateral elements and an example for the domain decomposition for 8 subdomains is shown in figure ??. The computational times, speedup and efficiency plots given in figures ??, ?? and ?? for the complex 'real life' example verify the results of the previous, more academic example. For higher polynomial degrees (p$\geq$ 8) and a small number of subdomains (1–3) the speedup is less than theoretically expected. This effect results from the imperfect load balancing of the domain decomposition. The subdomains are created only with respect to an equal number of elements, not considering the computational effort for columns. Yet, the effort for numerical integration of elements containing columns is much higher than for other elements, resulting in this imbalance for a low number of processors.

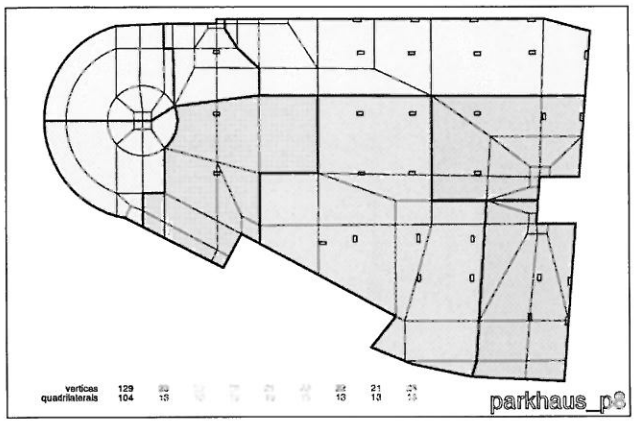

Figure 9: Car park – Finite element mesh and domain decomposition for 8 subdomains (by METIS [?])

p	dof	t_s [s]	memory [Mb]	t_2 [s]	t_4 [s]	t_8 [s]
6	11307	72.6	20.732	54.6	41.1	34.3
7	15375	119.9	28.702	83.8	55.6	45.9
8	20067	200.5	39.696	134.0	82.2	61.5
9	25383	350.5	55.216	245.7	127.3	90.6
10	31323	592.3	75.601	400.6	195.7	134.7
11	37887	1027.3	101.573	655.6	315.3	211.8
12	45075	1631.6	134.118	1011.5	497.5	318.0
13	52887	2588.3	174.306	–	758.7	475.0

Table 2: Degrees of freedom, time, and memory for the Car park example for the sequential computation and computational time for parallel computation with 2, 4, and 8 subdomains/processors

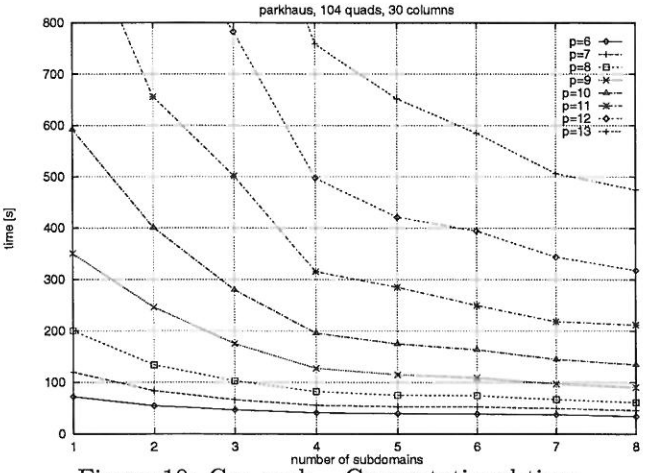

Figure 10: Car park – Computational time

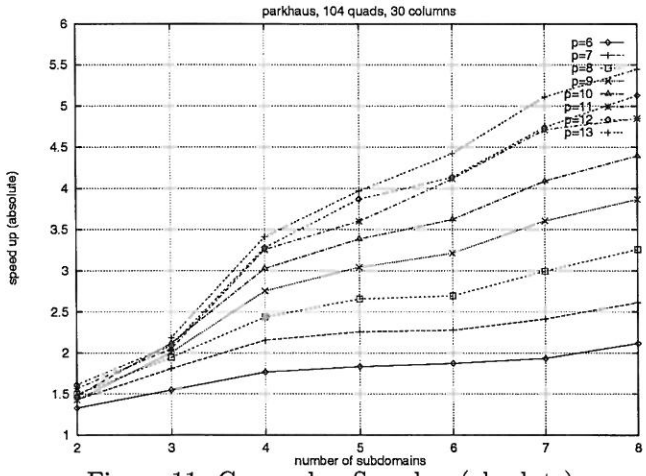

Figure 11: Car park – Speedup (absolute)

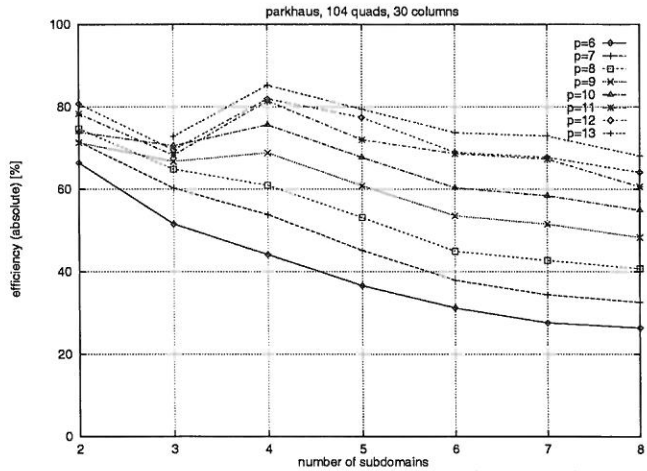

Figure 12: Car park – Efficiency (absolute)

References

[1] http://www.inf.bauwesen.tu-muenchen.de/projekte/inside/start.htm

[2] B. Szabó: "Mesh design for the p-version of the finite element method", Comp. Meth. in Applied Mech. and Eng., 55:181–197, 1986.

[3] B. Szabó and I. Babuška: Finite Element Analysis, Wiley, Chichester, 1990.

[4] W.J. Gordon and Ch.A. Hall: "Construction of Curvilinear Co-ordinate Systems and Applications to Mesh Generation", *International Journal for Numerical Methods in Engineering*, Vol. 7:461–477, 1973

[5] S. Bitzarakis, M. Papadrakakis, and D. Charmpis: "Parallel Solvers – Performance assessment", Esprit Project INSIDE, Report D51a, 1997.

[6] M. Ainsworth: "A preconditioner based on domain decomposition for hp-FE approximation on quasi-uniform meshes", SIAM J. Num. Anal., 33, No. 4: 1358–1376, 1996.

[7] http://www.epm.ornl.gov/pvm

[8] http://www-users.cs.umn.edu/ karypis/ metis/metis.html

[9] T. Ungerer: "Parallelrechner und parallele Programmierung", Spektrum Akademischer Verlag, Heidelberg, Berlin, 1997.

[10] E. Stein, K.H. Lambertz, L. Plank, and A. Reisch: "Element Benchmark Rhombusplatte 60°, Ergebnisse für: DKT-Dreieckselement, DKT-Viereckselement, Mindlin-Element mit reduziertem Schubansatz", Technical Report, Universität Hannover, Institut für Baumechanik und Numerische Mechanik, 1987.

[11] I. Babuška and T. Scapolla: "Benchmark computation and performance evaluation for a rhombic plate bending problem". International Journal for Numerical Methods in Engineering, 18:323–341, 1982.