Parallel Finite Element Methods with Weighted Linear B-Splines

Klaus Höllig, Jörg Hörner and Martina Pfeil

IMNG, University of Stuttgart, Pfaffenwaldring 57, 70569 Stuttgart Germany,
hoellig@mathematik.uni-stuttgart.de

Weighted extended B-splines (web-splines) combine the computational efficiency of B-splines and the geometric flexibility of standard finite elements on unstructured meshes. These new finite elements on uniform grids (cf. [5] and www.web-spline.de) are ideally suited for vectorization, parallelization and multilevel techniques.

In this project we explore the potential of the web-method for large scale applications with performance tests on the NEC SX-8 cluster of the HLRS. We implement a new minimal degree variant which uses predefined instruction sequences for matrix assembly and is almost as efficient as a difference scheme on rectangular domains.

1 Introduction

B-splines play an important role in approximation, numerical analysis, automated manufacturing, and computer graphics. Their use as finite elements suggests itself. However, in view of missing geometric flexibility and stability problems, this did not seem feasible. With the web-method, introduced in 2001 and patented in 2003, both difficulties have been overcome. Boundary conditions are incorporated via weight functions and stability is achieved by forming appropriate B-spline combinations. The resulting new type of a finite element basis, consisting of weighted extended B-splines (web-splines), combines the advantages of uniform splines and finite elements on unstructured grids:

- uniform grid with one basis function per grid point
- arbitrary smoothness and polynomial degree
- high accuracy with relatively few parameters
- exact fulfilment of essential boundary conditions
- hierarchical bases for adaptive refinements
- natural parallelism

The potential of the new method, which bridges the gap between geometric modeling and numerical simulation, becomes apparent from a variety of tests for model problems in elasticity, heat conduction, and fluid flow (cf. www.web-spline.de/examples/).

In this project we introduce and implement a minimal degree variant of the web-method for three-dimensional boundary value problems. It is intended primarily for simulations where computational speed is of key importance and only moderate accuracy is required (e.g. because of missing precision of the physical model, measurement errors, etc.). Poisson's equation as a basic model problem already exhibits the essential features of the new method, in particular, the treatment of domains with complicated boundaries via a special integration technique. The generalization to more general elliptic equations or systems is straightforward.

In this report we first describe in section 2 the weighted linear finite element basis. Section 3 is then devoted to a preprocessing technique which substantially accelerates the assembly of the Ritz-Galerkin system. A brief outline of the program structure is given in section 4. Finally, we illustrate in section 5 the performance of our algorithms.

2 Finite Element Basis

We choose scaled translates

$$b_k = b(\cdot/h - k), \quad (k_1, k_2, k_3) \in K = \{0, \ldots, 1/h\}^3,\ 1/h \in \mathbb{N}\,,$$

of a linear box-spline b [1] to construct a minimal degree web-basis (cf. figure 1). While the underlying partition, which consists of 6 tetrahedra S per grid cell $Q = \ell h + [0,1]^3 h$, is more complicated than for trilinear tensor product B-splines (the standard linear web-basis function), this slight disadvantage is outweighed by the smaller support and the smaller number of monomial coefficients.

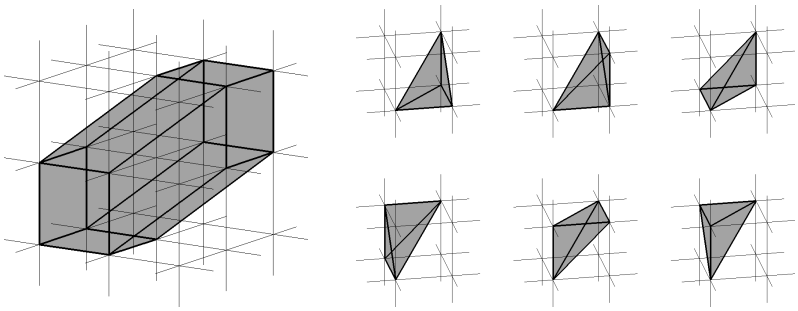


Fig. 1. Support and tetrahedral partition of a linear box-spline b_k

A spline on the standard domain $Q^* = (0,1)^3$ is a linear combination of scaled box-spline translates:

$$p^h = \sum_{k \in K} p_k b_k.$$

Since b_k is equal to 1 at the center kh of its support and vanishes at all other grid points $k'h$, the coefficients p_k coincide with the grid values $p(kh)$ of the spline. In particular, on each tetrahedron S of the partition, $p^h(x)$ can be computed via linear interpolation from its 4 values at the vertices of S.

We represent the simulation domain D in implicit form:

$$D : w(x) > 0\,,$$

where w is referred to as a weight function. Assuming for convenience that D is a subset of the standard cube Q^*, an approximate representation is given by $D^h : w^h(x) > 0$ with

$$w \approx w^h = \sum_{k \in K} w_k b_k, \quad w_k = w(kh)\,.$$

This yields a piecewise linear approximation of the boundary of order $O(h^2)$.

Functions u, which vanish on the boundary of D (homogeneous Dirichlet boundary conditions), are approximated by weighted splines

$$u^h = w^h p^h = \sum_{k \in K} p_k (w^h b_k) = \left(\sum_{k \in K} w_k b_k \right) \left(\sum_{k \in K} p_k b_k \right)\,,$$

where we set $p_k = 0$ if the support of b_k lies outside of D^h. In contrast to a standard finite element basis, consisting of hat functions (for example the

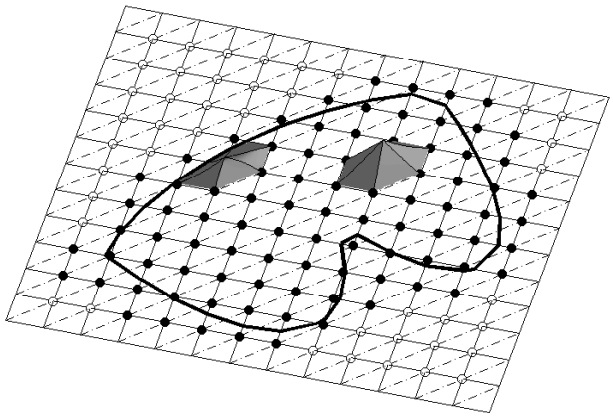


Fig. 2. Weighted linear B-spline basis for a two-dimensional domain

box-splines b_k), we use the weighted basis functions $w^h b_k$, thus conforming to boundary conditions despite the regular grid. Regardless of the shape of the domain, the weighted spline approximation is determined by the two arrays

$$w_k,\, p_k, \quad k \in K,$$

of dimension $(1 + 1/h)^3$. Moreover, on each tetrahedron S, u^h is a product of two linear functions which interpolate the 4×2 values of w^h and p^h at the vertices of S.

Figure 2 illustrates an analogous construction in two dimensions. The examples of several weighted bivariate linear B-splines show that the qualitative form of the new basis is quite similar to standard elements. The modifications appear to be relatively minor, yet the simplications are substantial. No grid generation is required, and the data structure is extremely simple.

3 Preprocessing of Ritz-Galerkin Integrals

The matrix entries of the Ritz-Galerkin system for Poisson's equation, which serves as our basic model problem, are

$$\int_{D^h} \operatorname{grad}(w^h b_k) \operatorname{grad}(w^h b_{k'}), \quad k, k' \in K\,.$$

These integrals are computed by summing the contributions from each grid cell Q:

$$\int_{D^h} \ldots = \sum_Q \int_{Q \cap D^h} \ldots\,. \tag{1}$$

The latter integrals are nonzero only if both, b_k and $b_{k'}$, have some support in Q, i.e., if kh and $k'h$ are vertices of Q. Hence, we can write

$$\int_{Q \cap D^h} \ldots = I(\ell, \alpha, \alpha', w^h), \quad \ell \in L = \{1, \ldots, 1/h\}^3,\, \alpha, \alpha' \in \{0,1\}^3,$$

where $Q = \ell h + [0,1]^3 h$, and $\alpha = k - \ell$, $\alpha' = k' - \ell$.

Usually, the Ritz-Galerkin integrals are computed on run-time. However, processing the boundary cells is time-consuming, since one has to take the intersection pattern of the tetrahedral partition with the boundary into account. There is a simple remedy. As is illustrated in figure 3, for each cell Q, the combinatorial type of the intersection of the tetrahedra $S \subset Q$ with the boundary of D^h is determined by the signature s of the values

$$W_\ell = \{w_{\ell+\gamma} : \gamma \in \{0,1\}^3\}$$

of the weight function at the vertices of Q. Hence, there are only $2^8 = 256$ possibilities. The different patterns partition the lattice points L into disjoint

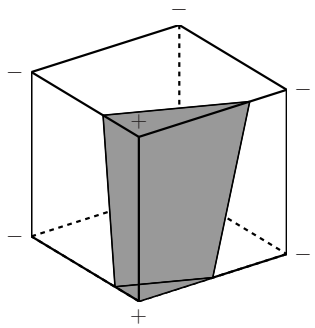


Fig. 3. Intersection pattern for a boundary cell, determined by the signature $\mathrm{sign}(w_\gamma), \gamma = (0,0,0), (1,0,0), \ldots, (1,1,1)$, of the weight function

sets L_s. For $\ell \in L_s$, the integral I is a rational function of the weight function values:

$$I(\ell, \alpha, \alpha', w^h) = R_{s,\alpha,\alpha'}(W_\ell), \quad \ell \in L_s\,.$$

Clearly, the functions R do not depend on the shape of the domain and can, therefore, be compiled as subroutines. This allows the complete vectorization of the computations for combinatorially equivalent cells, cutting the assembly time to a minimum.

Since the functions $R_{s,\alpha,\alpha'}$ are determined in advance, the time required for their automatic generation is irrelevant. Hence, we can optimize the instruction sequences for their evaluation. To find the best (or an almost best) procedure automatically is an interesting algebraic problem, which we are currently pursuing together with M. Clausen [2] from the University of Bonn. So far, we are using an extended version of the K-form algorithm of J. Koch [8] which is based on successive quadratic substitutions.

$$\frac{w_{0,0,0}^4 \left(2w_{0,0,0}^2 w_{1,1,0}^2 w_{1,1,1}^3 + 6w_{0,0,0}^4 w_{1,0,0} w_{1,1,1}^2 + \ldots \quad (107 \text{ similar terms})\right)}{(w_{0,0,0} - w_{1,0,0})^3 \, (w_{0,0,0} - w_{1,1,0})^2 \, (w_{0,0,0} - w_{1,0,1})^2 \, (w_{0,0,0} - w_{1,1,1})^2}$$

$$\left.\begin{array}{r} 4 \text{ substitutions of type } \tilde{w} = a/(b-c) \\ 17 \text{ substitutions of type } \tilde{w} = a \cdot b \\ 4 \text{ substitutions of type } \tilde{w} = a/b \end{array}\right\} \rightarrow \quad \text{simplified instruction set}$$

$$\begin{array}{l} 20\tilde{w}_5 - 6\tilde{w}_6 - \tilde{w}_7 - 10\tilde{w}_8 - 5\tilde{w}_9 - 10\tilde{w}_{10} - \tilde{w}_{11} - 5\tilde{w}_{12} + 10\tilde{w}_{13} + 2\tilde{w}_{14} + 2\tilde{w}_{15} \\ -2\tilde{w}_{16} - 8\tilde{w}_{17} + 4\tilde{w}_{18} - 2\tilde{w}_{19} + \tilde{w}_{20} + 2\tilde{w}_{21} + 4\tilde{w}_{22} + 2\tilde{w}_{23} + \tilde{w}_{24} + 2\tilde{w}_{25} \end{array}$$

Fig. 4. Example of a function $R_{s,\alpha,\alpha'}$ for Poisson's equation and a simplified instruction set

Figure 4 shows one of the $12384 = 256 \times 64$ expressions. We see that the evaluation can be considerably simplified. Further simplifications are possible, but not provided by the K-Form. Of course, given the number of expressions, such simplified instruction sets must be generated automatically.

4 Program Description

As for conventional finite element approximations, the numerical solution of an elliptic boundary value problem with weighted linear box-splines consists of two steps: the assembly of the Ritz-Galerkin system and its iterative solution. We describe each component of the algorithm in turn, considering only the more difficult matrix assembly.

The Ritz-Galerkin matrix has a constant (generalized) band-width. Accordingly, the entry (k, k') is denoted by

$$G(k, \Delta k), \quad k \in K,\ \Delta k = k' - k \in \{-1, 0, 1\}^3 .$$

To generate the array G, we first compute the cell integrals I. In general terms, this is accomplished by the following program segment.

```
for all s
    for all ℓ ∈ L_s
        for all α, α' ∈ {0,1}^3
            I(ℓ, α, α', w^h) = R_{s,α,α'}(W_ℓ)
        end
    end
end
```

Summation of the cell integrals according to (1) then yields the matrix entries, as described in the following program segment.

```
G = 0
for all α, α' ∈ {0,1}^3
    for all ℓ ∈ L
        G(ℓ + α, α' − α) = G(ℓ + α, α' − α) + I(ℓ, α, α', w^h)
    end
end
```

In addition, we set zero diagonal entries $G(k, 0)$ equal to one, in order to keep a simple data structure regardless of the shape of the domain. The corresponding entries of the right-hand side are set to 0. This is in agreement with the convention that the box-spline coefficients p_k are zero for $D^h \cap \operatorname{supp} b_k = \emptyset$.

The notation and index structure of the pseudo-code reflects the mathematical description, not the FORTRAN implementation. However, it is apparent that the main loops are easily vectorized.

To solve the Ritz-Galerkin system, we use a dynamic version of the multigrid algorithm, described in [6]. The two basic components, Richardson smoothing and grid transfer with box-spline subdivision, trivially vectorize over the regular grid. In fact, the data structure of web-splines is ideally suited for any type of iterative solver.

The simplicity of the above program fragments is deceptive – the complexity of the problem must be hidden somewhere. It is contained in the subroutines for the functions $R_{s,\alpha,\alpha'}$. The corresponding automatically generated code (244,582 lines) constitutes with approximately 98% the major portion of the program.

5 Implementation and Performance

As a first test, we implemented a solver for the 2-dimensional Poisson equation. The code is written FORTAN90 with the following key routines:

- CELL_DATA: evaluation of the functions $R_{s,\alpha,\alpha'}$;
- MATRIX: assembly of the Ritz-Galerkin matrix;
- MG_SMOOTH: basic multigrid iteration.

The first routine, comprising the major portion of the code, is automatically generated with the aid of a MATLAB-program. It not only supplies instruction sequences for integration, but also optimizes the complexity with K-form substitutions. Richardson's iteration is used as a smoother, which in our experiments proved to be superior to SSOR and checkerboard-SSOR.

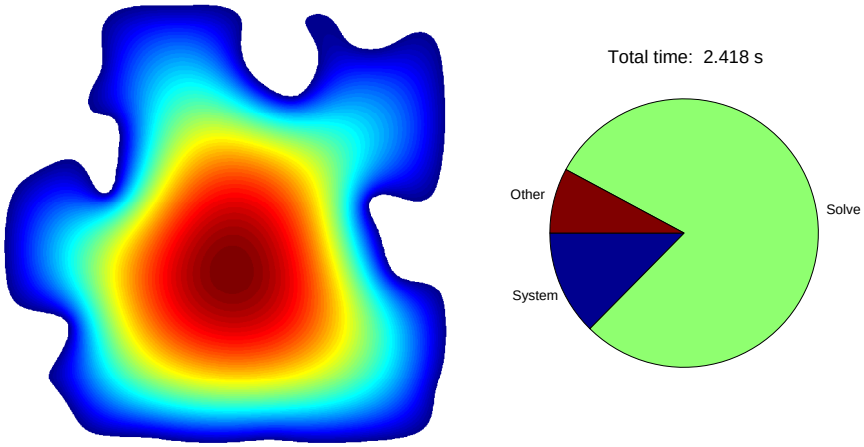


Fig. 5. Solution and computing times for a 2-dimensional domain

An advantage of our new meshless method is the simplicity of the boundary treatment. Therefore, and in order to compare with automatic grid generation, we use random domains with fairly irregular boundaries. For the example shown in figure 5, a system with more than 6 million unknowns was solved on one processor in less than 2.5 seconds. As is typical for our B-spline based method, the assembly time (which corresponds to grid generation and numerical integration for a standard finite element method) is cut down substantially due to the regular grid and the use of the predefined instruction sets (routine: CELL_DATA).

Table 1 gives more detailed information about the program performance. We see that for all three routines the average vector length is larger than 200, i.e. very close to the maximal length 256. As a consequence, the efficiency is almost optimal. For the matrix assembly (routine: MATRIX) 7.9 Gflops are reached. The performance of the Richardson iteration with 6.1 Gflops is also quite good, in particular, since most of the 1204 iterations operate on coarse grids. Using a count weighted with the grid size, the solver requires 9.684 iterations in fine grid units. This roughly corresponds to an error reduction by a factor 0.17 per multigrid cycle.

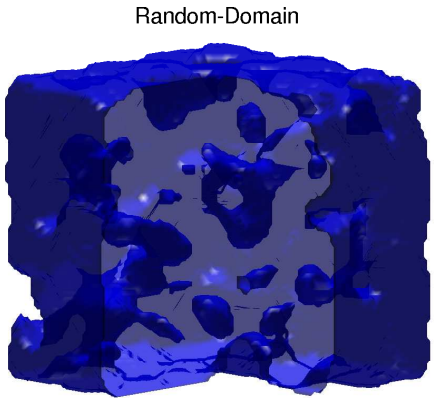
The routines, not listed in the statistic, also vectorize well (the vector ratio never drops below 98%); yet they contribute only a small fraction ($< 14\%$) to the total time.

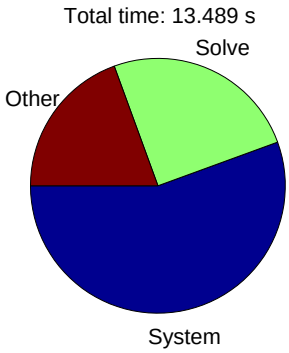
PROG_UNIT	FREQ.	TIME	MFLOPS	V.RATIO	V.LEN
CELL_DATA	9	0.196	23.0	98.24	230.3
MATRIX	9	0.096	7905.4	99.76	245.2
MG_SMOOTH	1204	1.794	6173.1	99.40	211.8

Table 1. Statistics (ftrace) of the main routines for the two-dimensional test case (V.RATIO = ratio of vector operations, V.LEN = average vector length)

The full potential of the new algorithm becomes apparent when considering three-dimensional examples. In fact, the random domain, shown in figure 6, would be nontrivial to mesh, as is necessary for a standard finite element scheme. The performance is not quite as good as for the two-dimensional case. While we achieve > 7.2 and > 4.6 Gflops for the routines MATRIX and MG_SMOOTH, the routine CELL_DATA (here split according to the different cell types into routines CELL_DATA_K, K=001:254) does not vectorize well. The reason is that not all of the 254 boundary intersection patterns occur frequently enough. Of course, for smaller grid-width, the vectorization would be close to optimal again.

It is conceivable that significant improvements of the performance are still possible. First, our code can almost certainly be tuned further to the specific architecture. Second, the K-form can be augmented by additional algebraic





PROG_UNIT	FREQUENCY	TIME	MFLOPS	V.RATIO	V.LEN
MATRIX	5	0.273	7210.1	99.70	124.1
MG_SMOOTH	499	3.011	4640.9	97.99	98.1
CELL_DATA_240	5	0.267	313.1	45.91	51.6
CELL_DATA_085	5	0.264	310.2	48.97	54.8
⋮					
CELL_DATA_186	5	0.024	162.5	86.18	138.1

Fig. 6. Domain and statistics for a 3-dimensional test case

simplification tools. In fact, it is an intriguing mathematical problem to automatically determine the best evaluation sequence for the functions $R_{s,\alpha,\alpha'}$.

6 Concluding Remarks

The numerical tests have confirmed that our new method is well suited for handling complicated boundaries. Moreover, due to the regular grid and an extremely simple data structure, the vectorization has led to a considerable speed up of our serial implementation (factor 15 in comparison to a 2.4 GHz AMD-CPU). The efficiency is partially also due to the idea of using predefined instruction sets, which avoids spending time on calculating quadrature points for boundary cells.

The generalization to variable coefficients problems and other elliptic problems, such as the Navier-Lame system of linear elasticity and eigenvalue problems for water waves, is relatively straightforward. It would also be interesting to apply the B-spline approach to moving boundary problems and form optimization. In both cases, the weight function technique could prove to be an alternative to existing methods, which require remeshing or have some topological limitations.

Acknowledgement: We thank Dr. U. Küster and the HLRS parallel computing group who, with their advice and excellent support, contributed to a considerable percentage of the Gflops mentioned above.

References

1. C. de Boor, K. Höllig, and S. Riemenschneider: *Box Splines,* Springer-Verlag, New York, 1993.
2. M. Clausen, Private communication, 2007.
3. K. Höllig, *Finite Element Methods with B-Splines,* SIAM, 2003.
4. K. Höllig, U. Reif, and J. Wipper: *Weighted extended B-spline approximation of Dirichlet problems,* SIAM J. Numer. Anal. 39 (2001), 442–462.
5. K. Höllig, U. Reif, and J. Wipper: *Verfahren zur Erhöhung der Leistungsfähigkeit einer Computereinrichtung bei Finite-Elemente-Simulationen und eine solche Computereinrichtung,* Deutsches Patent DE 100 23 377 C2 (2003).
6. K. Höllig, U. Reif, and J. Wipper: *Multigrid methods with web-splines,* Numer. Math. 91 (2002), 237–256.
7. J. Hörner: MIND: Multiple integration over NURBS domains, 2007.
8. J. Koch: *Subdivision-Algorithmen zur Lösung polynomialer Gleichungssysteme,* Master's Thesis, University of Stuttgart (1991)
9. Finite Element Approximation with WEB-Splines (Slide collection): http://www.web-spline.de/publications/slide_collection.pdf