Efficient FEM implementation

The finite element method (FEM) is used to solve partial differential equations (PDE). The key element in this method is that it uses a variational problem over a domain of the PDE. The weak form of a PDE has a test and a shape function which enables to discretize the system. The evaluation of the integrand produces a system of equations. For the purpose of this document only the efficiency of linear systems of equations, obtained from FEM, is treated. Approaches for linearizing non-linear system of equations can be done with other numerical methods.
FEM produces a linear system of equations in the form

$$A \cdot u = b \tag{0}$$

where
A is the system matrix,
u is a vector with the unknowns representing vertices in a Cartesian grid and
b is a vector containing the right hand side values.

The system matrix A is typically sparse and has a regular structure. The structure of the rows of A corresponds to the stencil of the nodes in a Cartesian grid.

Cartesian grids usually need high resolution in specific areas. The need for the use of an adaptive grid arises with:

- Complex geometry boundaries: detailed observation of the solution of geometric features.
- Singularities: very local and relatively high changes in the solution of a PDE. Adaptivity is related to discretization error. The refinement may done during runtime and not a priori.
- Multi-scale phenomena: local changes or small scale effects are of interest in the solution of a PDE.

The refinement is done in sub domains (where high resolution is needed) in a recursive way, by splitting a cell into sub grids.

The element wise view of a stencil is done by separating each vertex into cells. Instead of looking at the vertices we examine the elements considering partial values of the vertices. For example the stencil:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Can be represented in the element wise view as follows:

$$\begin{bmatrix} -1 & -1/2 \\ -1/2 & 2 \end{bmatrix} + \begin{bmatrix} -1/2 & -1 \\ 2 & -1/2 \end{bmatrix} + \begin{bmatrix} -1/2 & 2 \\ -1 & -1/2 \end{bmatrix} + \begin{bmatrix} 2 & -1/2 \\ -1/2 & -1 \end{bmatrix}$$

We can profit from the element-wise view when applying space filling curves in adaptive grids. We use data structures like stacks that are processed faster than other accesses to data structures, thus improving computing time.

An implicit solver like the Jacobi's iterative method may be used to solve a large system of equations. This document focuses on the Jacobi's solver, but other methods may be used. Others widely used are Gauss-Seidel and Conjugate Gradient. Multigrid method is also used combined to increase the efficiency of a given solver.

The general formulation of the Jacobi solver is as follows:

$$u^{(i+1)} = u^{(i)} + \frac{1}{\mathrm{diag}(A)}(b - Au^{(i)}) \tag{1}$$

$$u^{(i+1)} = u^{(i)} + \frac{1}{\mathrm{diag}(A)}r^{(i)} \tag{2}$$

Where
Index i is the iterative step
A is the system matrix,
u is a vector with the unknowns representing vertices in a Cartesian grid and
b is a vector containing the right hand side values
diag(A) is a vector containing the main diagonal of A.

Jacobi basic algorithm visits all "k" unknowns as follows:

Do while residual is sufficiently small:

$$\text{for } k = 0, 1, ..., n-1 \qquad \omega^{(i)} = \frac{1}{a}r_k^{(i)} \tag{3}$$

$$\text{for } k = 0, 1, ..., n-1 \qquad u^{(i+1)} = u_k^{(i)} + \omega_k^{(i)} \tag{4}$$

End While.

A common problem encountered in high performance computing is how to profit from cache efficiency. Cache misses are expensive in terms of time and we require algorithms to minimize them. The goal is to ensure that the information referenced at one point in time will be referenced in the near future. A second aim is to benefit from the temporal locality. When visiting all the elements on a grid the processor will load vertex data that was already loaded into the cache line.

The Peano curve will traverse the grid in a characteristic order, which reduces the cache misses. The grid will be traversed only once and the cells visited in a succession will be neighbouring cells. Other space filling curves may be used, e.g. Hilbert curve and Sierpinsky curve.

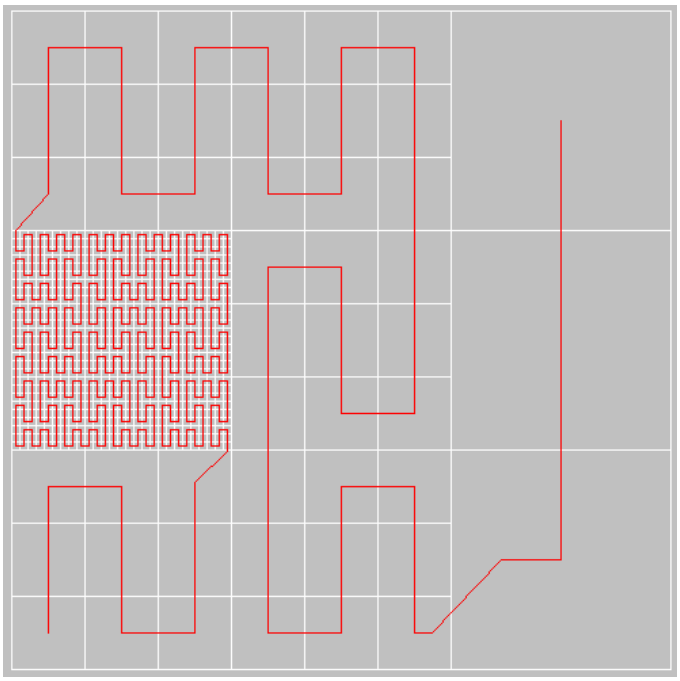

Figure 1: Peano Curve with different levels of resolution

The space filling curve now needs to be represented with a scheme suitable for a computer program. This representation as a data structure can be done with a tree. The data of a cell will be stored at the corresponding node. The root represents the lowest level of resolution and the leaves the high resolution levels. Thus, the grid is traversed in a top-down-depth order.

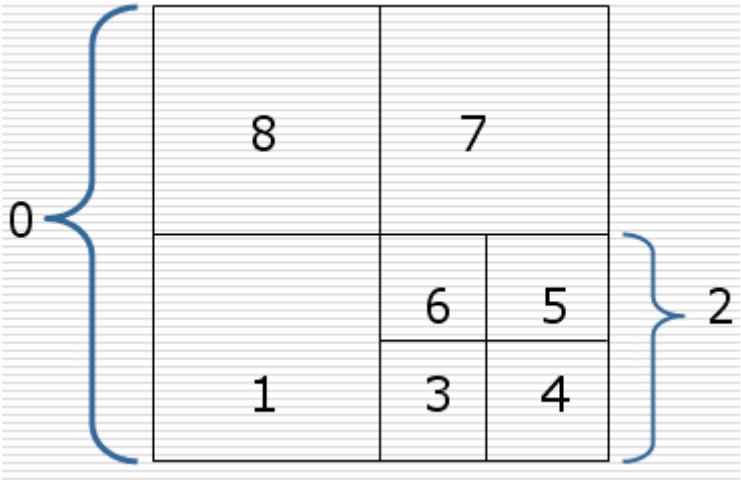

Figure 2: Simple adaptive grid with numbering

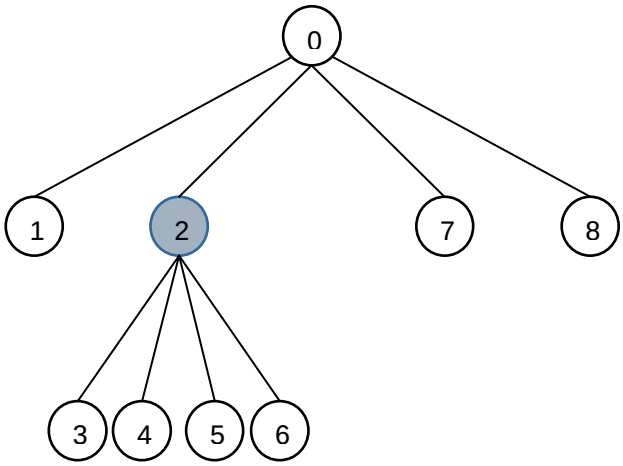

Figure 3: Tree structure that represents the grid in figure 2.

Notice in figure 3 that every element will contain one refinement bit attached to it. If the element‘s refinement bit is set, the node has children and these are visited.

If it is possible to parallelize the process to pursue a faster computing time, we should consider the following: communication between processors is slow and therefore sub boundary domains should be minimized. This leads to form compact domains with a low ration communication surface and volume. The balancing of domains in adaptive grids is not straight forward. When adaptivity occurs during runtime, the parallel adaptive grid control needs dynamic load balancing. The amount of workload per processor should be roughly the total amount of elements divided by the number of processors.

References:

1. Günther, F.; Mehl, M.; Pögl M.; and Zenger C. *A Cache-Aware Algorithm for PDEs on Hierarchical Data Structures.* TUM paper.
2. Frank, A. *Organisationsprinzipien zur Integration von geometrischer Modellierung, numerische Simulation und Visualisierung.* TUM paper.
3. Mehl, M. and Zenger C.; *Cache-oblivious parallel multigrid solvers on adaptively refined grids.* TUM paper.
4. Bungartz, H-J.; Mehl, M.; and Weinzierl, T.; *A Parallel Adaptive Cartesian PDE Solver Using Space–Filling Curves.* TUM paper.