GdfidL ON CLUSTERS OF WORKSTATIONS

W. Bruns and H. Büssing, TU-Berlin, Berlin

Abstract

The electromagnetic field solver GdfidL has been extended to run efficiently on loosely coupled parallel systems, such as clusters of workstations. A computational volume which is discretised using a regular grid can be easily partitioned such that each processor has the same number of gridcells. If for every gridcell the needed computation is the same, such a partitioning is very efficient. However, for typical accelerator components, most gridcells are filled with metal, therefore no fields need to be computed in these cells. When such a geometry is parallelised naively, the computational efficiency goes down to about 10%, since most processors work on electrical conducting cells. In that case, the problem is finding a partitioning such, that each processor gets the same number of gridcells filled with vacuum. The paper describes the implemented algorithms for partitioning the computational volume and for computing the electromagnetic fields. The total achieved computational efficiency is about 70%.

1 THE PROBLEM

The Finite Difference Method in cartesian coordinates is easily parallelised, since the subdivision of the total rectangular computational volume is easily done, if one restricts oneself to rectangular subvolumes. One just has to partition the grid such, that each processor has about the same number of gridcells. This approach works well, when electromagnetic fields can exist in a large fraction of the volume.

However, most realistic RF-devices, if computed in a rectangular volume, do not lead to a grid where most gridcells are filled with vacuum or a dielectric. The opposite is the case: Complicated devices, for which the computation inherently is time consuming, have an enclosing rectangular box of which 90% or more is filled with electric conducting material. If one subdivides such a volume into as many subvolumes as there are processors, most processors will work on parts of the volume where the fields are known to be zero a priori. What a horrible waste of resources.

2 THE WAY OUT

Finite Element based codes do not have that problem. Since they anyway have to deal with their complicated mesh-topology, dealing with non-rectangular sub-volumes is their daily bread. But if one wants to stick with the inherently easier implementation and cheaper execution of classical Finite Differences, one has to stick to rectangular subvolumes.

There is a way out: You won't find in the ten commandments, that each processor is limited to working on a single subvolume. If we subdivide the total volume in many more subvolumes than we have processors, then we can discard the subvolumes where no fields need to be computed, and spread the remaining ones evenly over the available processors. This approach is halfway between classical Finite Difference and the complicated topology of Finite Element Meshes.

For a first, typical example, in figure 1 we present a model of the BESSY cavity.

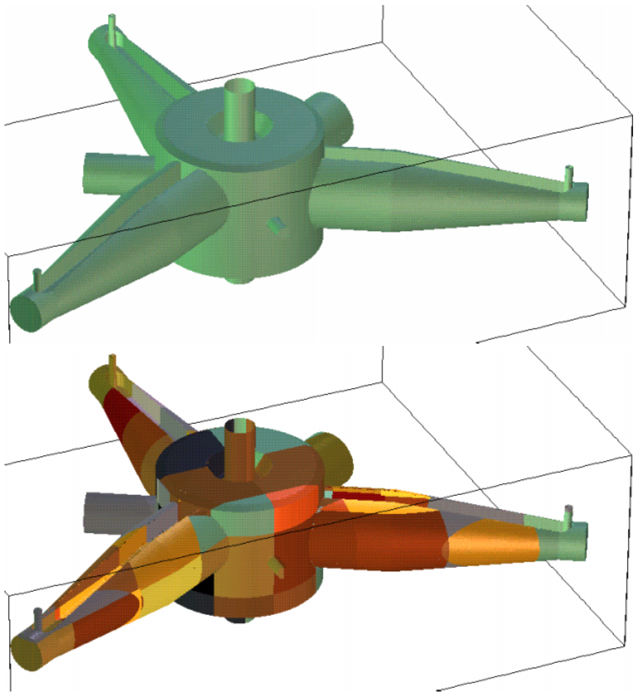

Figure 1: Above: A model of the BESSY cavity with attached waveguides and tuning plungers. The three large damping waveguides are attached at the cavity at different heights, therefore no symmetry plane is left. Because of the large damping waveguides, most of the enclosing box is filled with electric cells. In this case, less than 10% of the computational volume is filled with vacuum cells. The total number of grid cells used is about 16 millions. Below: The same model with different colours, indicating the used subvolumes. The total volume is subdivided in 8x24=192 subvolumes, of which 122 can be discarded, since they do not have a single vacuum cell. GdfidL needs about 3 GBytes of RAM and six hours wall clock time on an eight processor PC Cluster (total cost 8.000 EUR) to accurately compute the first 120 resonant fields in that structure.

3 THE IMPLEMENTATION

With this concept, the implementation is straightforward, however tedius.

3.1 *Local field computation*

The core of the Finite Difference Method is the discretisation of the curl-operators. With these discretised curl operators, one computes time dependent fields (FDTD) via the discretised form of

$$\vec{H}(n\Delta t) = \vec{H}((n-1)\Delta t) - \Delta t \frac{1}{\mu} \nabla \times \vec{E}((n-1/2)\Delta t) \quad (1)$$

$$\vec{E}((n+1/2)\Delta t) = \vec{E}((n-1/2)\Delta t) + \Delta t \frac{1}{\varepsilon} \nabla \times \vec{H}(n\Delta t) \quad (2)$$

and one finds resonant fields by searching for the eigenvalues of the discretised form of

$$\frac{1}{\varepsilon} \nabla \times \frac{1}{\mu} \nabla \times \vec{E} = \omega^2 \vec{E} \quad (3)$$

Most of the CPU-time is spent in applying these discretised curl operators. However, they are quite easily parallelised. For example, when performing a FDTD calculation, the algorithm for a single subvolume per processor is:

```
For all Timesteps: DO
  Compute local H by applying the local
  curl operator to the local E
    For all Directions: DO
      Send tangential H to the neighbour
      Receive tangential H from the neighbour
    ENDDO For all Directions
  Compute local E
    For all Directions: DO
      Send tangential E to the neighbour
      Receive tangential E from the neighbour
    ENDDO For all Directions
ENDDO For all Timesteps
```

The tangential H-components at the lower boundaries of the local volumes must be sent to the neighbour volumes in negative directions. Correspondingly, the tangential E-components at the upper boundaries must be sent to the neighbour volumes in positive directions. For correct results, the tangential components from a neighbour in eg. x-direction must be received before data can be sent in eg. y-direction.

However, if one wants to have only a single thread of execution, the field update for more than one subvolume per processor must be done slightly more complicated, since otherwise deadlocks will occur:

```
For all Timesteps: DO
  For all Subvolumes: DO
    Compute local H by applying the local
    curl operator to the local E
  ENDDO For all Subvolumes
  For all Subvolumes: DO
    For all Directions: DO
      Send tangential H to the neighbour
      Receive tangential H from the neighbour
    ENDDO For all Directions
  ENDDO For all Subvolumes
  For all Subvolumes: DO
    Compute local E
  ENDDO For all Subvolumes
  For all Subvolumes: DO
    For all Directions: DO
      Send tangential E to the neighbour
      Receive tangential E from the neighbour
    ENDDO For all Directions
  ENDDO For all Subvolumes
ENDDO For all Timesteps
```

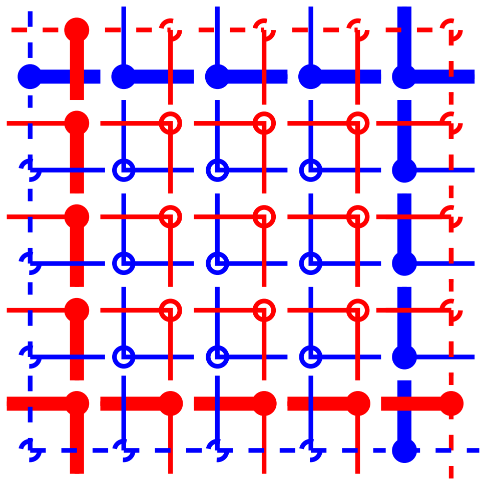

Figure 2: The blue lines and circles represent the electric field components in a local volume. The red ones represent the magnetic field components. The tangential E field components at the upper boundaries of the local volume (thick blue) and the tangential H field components at the lower boundaries (thick red) can be computed from the local information. These components are sent to the neighbour volumes. The tangential E field at the lower boundaries (dashed blue) and the tangential H field at the upper boundaries (dashed red) cannot be computed from the local information. These components are received from the neighbours.

3.2 *Grid Generation*

It is crucial that the grid generation is parallelised as well, otherwise that step would be the most time consuming part. The generation of the grid and the FD-coefficients is made via two passes: In the first pass, each processor gets assigned the same number of subvolumes to generate the coefficients for. After that first pass, each subvolume is inspected, how many gridcells can carry a nonzero field. The subvolumes with only zero field are discarded, and the remaining ones are spread over the available processor such, that a: each processor has about the same number of interesting grid-cells to deal with, and b: that the communication between subvolumes on different processors is minimised. This spreading over the available processors is implemented via a call of the METIS[1] package. In the second pass, each processor generates the mesh and the coefficients for the subvolumes which were assigned to him.

4 THE ACHIEVED EFFICIENCY

The parallelisation with many subvolumes is implemented in GdfidL[2] for the computation of eigenvalues with and without periodic boundary conditions, for the computation of scattering parameter computations, and for wakepotential computations. In all cases, lossy and/ or dispersive materials are allowed.

4.1 Definition of Efficiency

A cluster of N loosely coupled computers (with a cost of N times more than 1 computer) has N times the memory, and has N times the CPU power of a single computer. The efficiency is the utilisation of these resources.

- Largest possible problem: If one can use grids with $(f_{RAM} \times N)$ times more gridcells, before the RAM of one of the computers is exhausted, one has a memory efficiency of f_{RAM}.
- Faster execution: If one can compute a given problem in a time which is smaller by a factor of $f_{CPU} \times N$ than the time on a single computer, one has a CPU efficiency of f_{CPU}.

For moderately large number of gridcells, say more than 1 Million per computer, both, the memory efficiency and the CPU efficiency go up with the number of subvolumes, since more subvolumes can be more evenly spread over the processors. Since the subvolumes overlap slightly, the efficiencies will eventually decrease for extremely many subvolumes. The CPU efficency additionally decreases for many subvolumes, since the required communication is proportional to the surface of the subvolumes. The measured efficiencies on a cluster of 8 processors, connected via cheap 100 Mbit ethernet, are about 80% for the memory efficiency, and 70% for the CPU efficiency.

As a second example, figure 3 presents the model of the Cornell cavity.

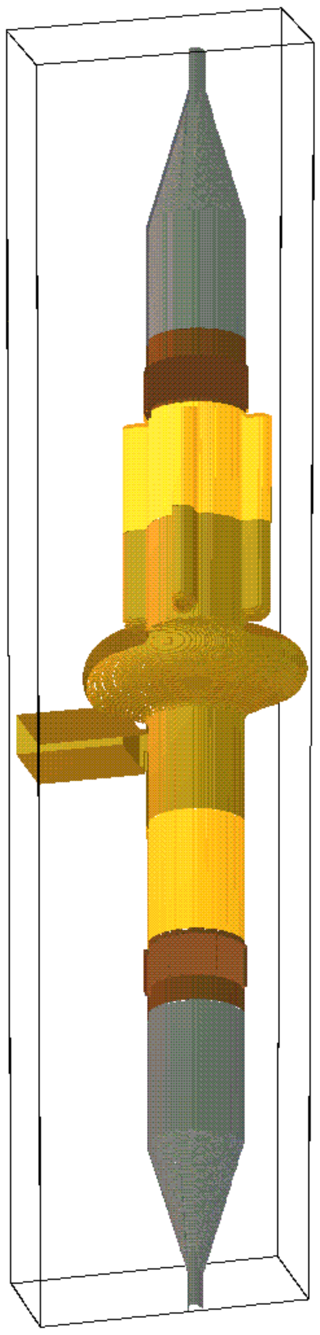

Figure 3: A model of the Cornell-Cavity, with attached feeding waveguide and with absorbers at the circumference of the beampipe. The total number of used subvolumes is 96, of which 52 can be discarded, since they are fully filled with electric conducting material. A long range wakepotential computation up to s=10 meters (three times the structure length) in a grid of 140 million cells needs about 1.7 GBytes of RAM and four hours of wall clock time on an eight processor PC Cluster.

5 REFERENCES

[1] http://www-users.cs.umn.edu/∼karypis/metis/

[2] http://www.gdfidl.de