A Parallelisation Approach for Multi-Resolution Grids Based Upon the Peano Space-Filling Curve

Usually in solving real life problems (e.g. Fluid Flow Simulation) we have the following stages:

- modelling physical problem (e.g. Fluid Flow)
- discretise the domain (e.g. the container in which the fluid flows)
- solve the discretised problem.

For study purpose we will take a very simple example (squared domain) but in real life problems we have more complicated domains (containing obstacles etc.). By discretising the domain we obtain a grid and for each node we have to compute data (e.g. pressure, velocity) depending on the data contained in the current node as well as on the data contained in the neighbouring nodes.

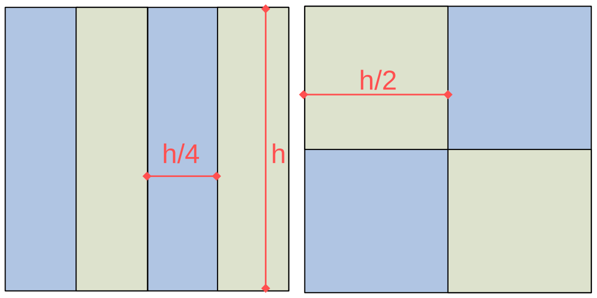
If we have a big domain that is discretised and we obtain many nodes we have a lot of computations to do. As in our days processors are not getting faster multicore processors and supercomputers are the solution for decreasing the computing time.

But having to compute values for every node that depend on the neighbouring nodes means that we have to take into consideration the data that has to be communicated and to find strategies to divide our domain in order to be parallelised. In this paper we will try to give a solution for parallelising the domain using the Peano Space-Filling Curve.

1. Domain discretisation

<u>Def:</u> <u>Domain partitioning</u> is the process of dividing a domain into two or more regions.

We have more that one opportunity to split a domain. Let's take the example of splitting the domain into rectangular or in squared regions.



Which of the two partitioning is better? One way of measuring the quality of a partition is to take the ratio between the communicated data and the local data. The lower this ratio is the better partitioning we obtain, because we want for each processor to have more data computing and less data communication.

This ratio is translated in 2D as the ratio between the length of the element divided by its area and in 3D as the surface over the volume.

$$Partition\ quality = \frac{communicated\ data}{local\ data} \overset{2D}{=} \frac{length}{surface} \overset{3D}{=} \frac{surface}{volume}$$

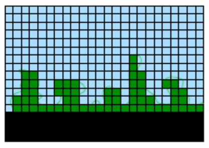
$$Partition\ quality_{stripes} = \frac{2\left(h + \frac{h}{4}\right)}{h \cdot \frac{h}{4}} = \frac{10}{h}$$

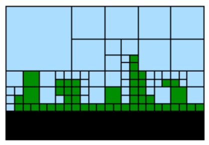
$$Partition\ quality_{square} = \frac{2\left(\frac{h}{2} + \frac{h}{2}\right)}{\frac{h}{2} \cdot \frac{h}{2}} = \frac{8}{h}$$

For our 2D small case we obtain for the rectangular partitioning a ratio of 10/h and for the squared one 8/h. As expected, the partitioning using rectangular areas is worse than the one using squares.

A normal question arises now: Which would be the lower limit? How good can our partitioning become? We have to find the geometrical figure with the smallest length and the biggest area. This is of course the circle.

In this talk, we want to restrict ourselves to two types of grids:


regular grids [4]


adaptive grids [4]

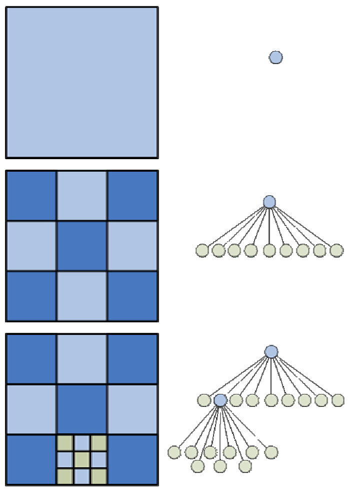
Comparison Table

	Regular Grid	**Adaptive Grid**
Generation	Easy	Difficult
Storage Scheme	Easy	Complicated
Algorithm Implementation	Easy	Difficult
Precision	Great memory and computational effort	Good

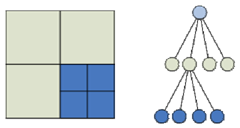
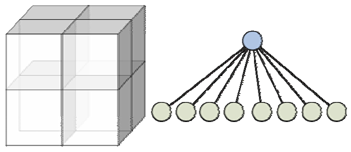
As we can observe from the table we would choose adaptive grids because of their property of helping in getting precise results with less amount of computational effort and also reduced memory usage. But we need to find a way to efficiently store in memory and also to develop methods to work with the data structure.

2. Computer representation – Trees

Often adaptive grids are constructed in a recursive manner, we need to find a data structure that suits it. We have different data structures for modelling this recursivity e.g. trees. In this paper we choose the tree structure as data structure.



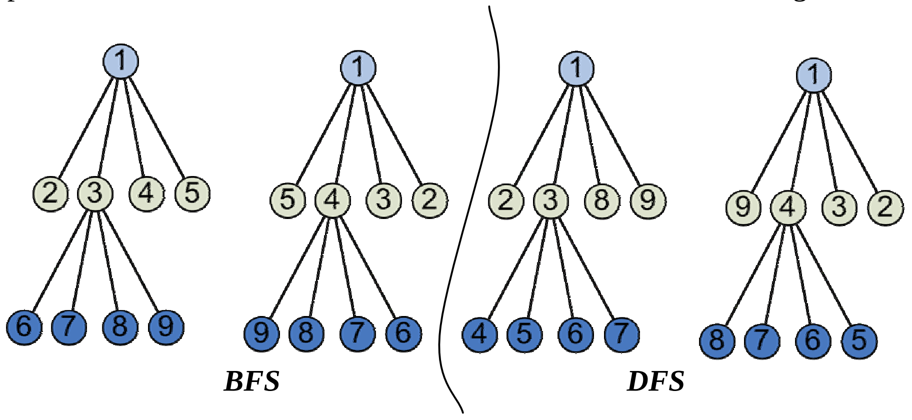
In literature we encounter different names for the tree data structure:

- for domain bipartition
    - 2D quadtrees 
    - 3D octrees 
- space trees
- refinement trees

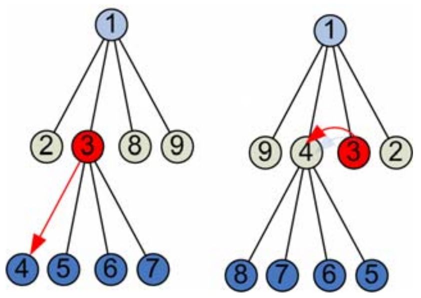
Now that we have the data structure, we need to know how to read from that structure. For the tree data structure we have basically the two traversal algorithms BFS (Breadth First Search) and DFS (Depth First Search).

BFS is a graph traversal algorithm that has the following strategy: starts at a given vertex, which is at level 0. In the first stage, we visit all vertices at level 1. In the second stage, we visit all vertices at second level. These new vertices, which are adjacent to level 1 vertices, and so on. The BFS traversal terminates when every vertex has been visited.

DFS is a graph traversal algorithm that has the following strategy: first visit the first child, then visit the first child of it and so on. When we reach a leaf (node without children) we go one step back and visit the next child of the node and so on [backtracking].



Our desire would be to have a deterministic algorithm i.e. at each moment in time the algorithm knows what is the next step to take. Just using a DFS traversal is not enough:

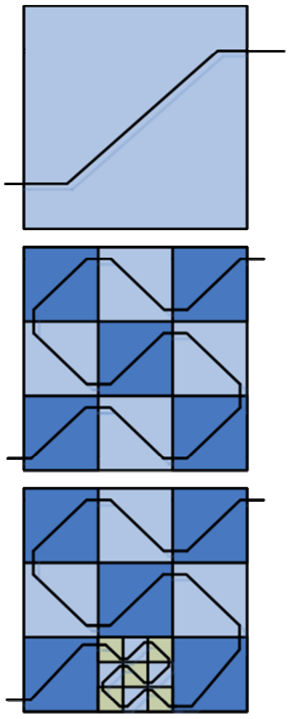


We need an order for our tree.

3. Traversal – Space-Filling Curves

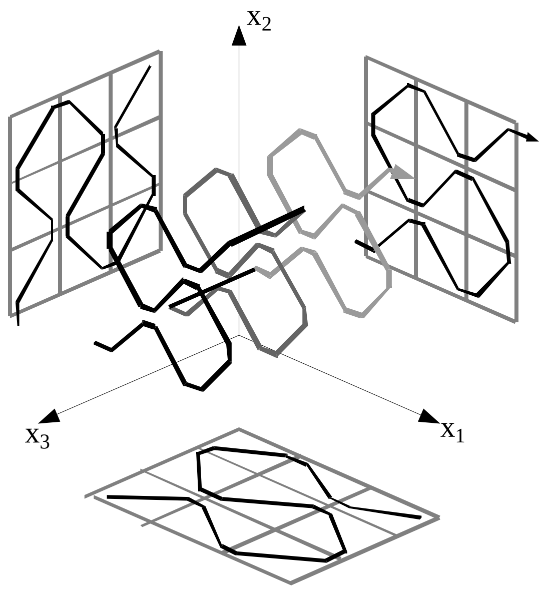
An N-dimensional space-filling curve is a continuous, surjective (onto) function from the unit interval [0,1] to the N-dimensional unit hypercube $[0,1]^N$. In particular, a 2-dimensional space-filling curve is a continuous curve that passes through every point of the unit square $[0,1]^2$. [5]

In this paper we choose the Peano Space-Filling Curve for tripartitioned adaptive grids to define the order of the children.

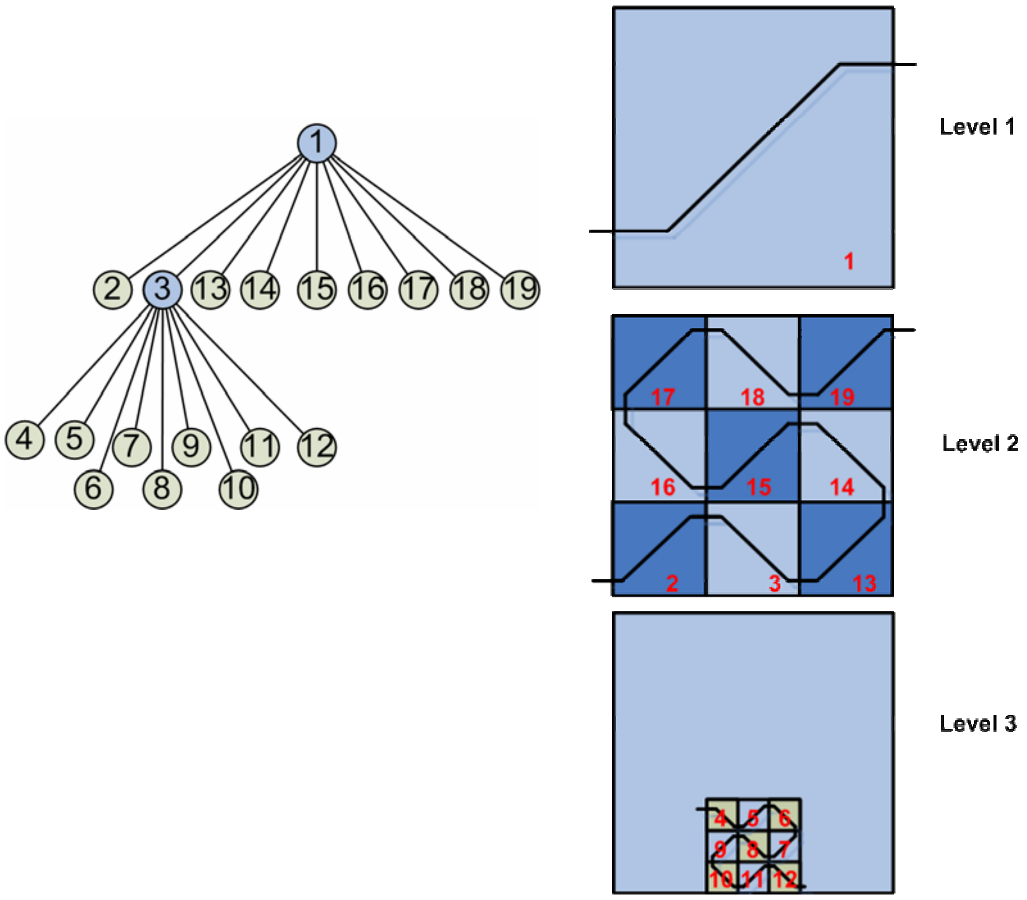


Peano Space Filling Curve for our grid.

We have chosen the Peano Space Filling Curve because of it's 3D projection propriety i.e. every projection on to the normal subplanes is a Peano Space Filling Curve again [2]:



For our tree we obtain a traversal given by the Peano Space-Filling Curve as follows:



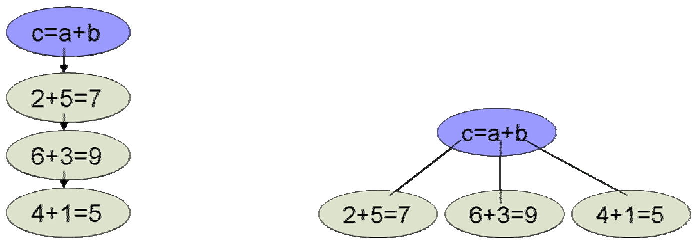
4. Parallelisation

Having chosen our types of grids (adaptive grids), an efficient storage scheme (tees), traversal algorithm (DFS) and last but not least an unique order for the children (Peano Space-Filling Curve) we can say that we obtained all the prerequisites for the parallelisation of our algorithm.

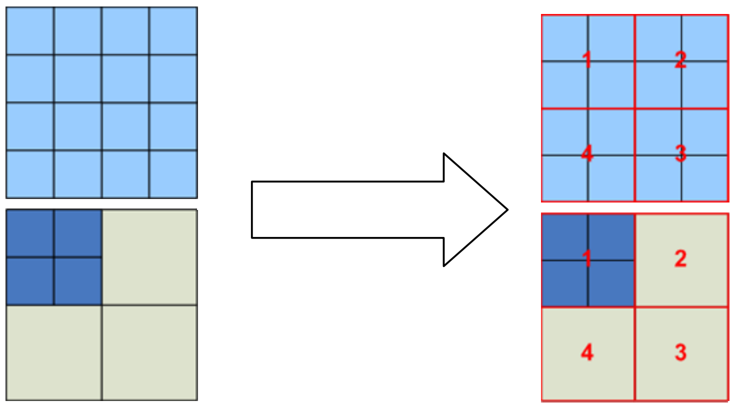
Def Parallel computing is the division of one task into a set of subtasks assigned to multiple processors in order to obtain results faster or to reduce memory requirements per processor.

Simple example: *input:* a={2,3,4}, b={5,6,1}
result: c=a+b={7,9,5}

Serial Parallel (3 processors)



Remember our two different structured grids: regular and adaptive. We will try now to divide the work between four processors using our square partitioning:



Example of domain partitioning of a regular grid (up) and adaptive grid(down) for parallelising for four processors (numbered 1,2,3,4, right)

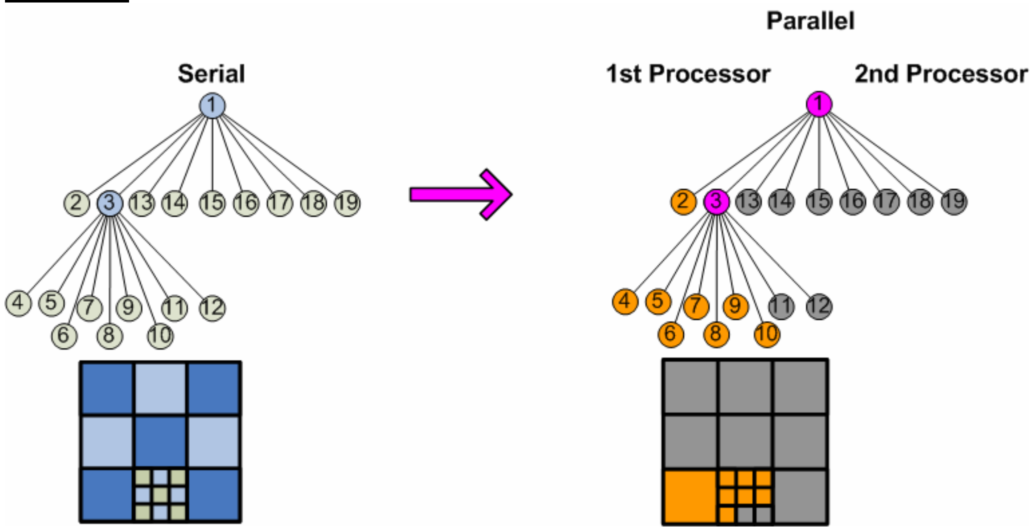
Well, it is a partition, but we can say just by looking at it that it works better for regular grids. For adaptive grids we have an unbalanced workload. Processor 1 has to do four times more work than the other three thus increasing the total computing time because the other processors have to wait for the first one to finish.

Def: Load balancing is a technique to spread work between processors in order to get a good resource utilisation and, thus, decreasing computing time.

This parallelisation strategy of dividing the grid into squared domains is not a very efficient one as we have seen so, we need another strategy. One strategy is introduced by Mitchell in [1]. Instead of splitting the grid we split the tree attached to the grid. If we have p processors and n nodes in our tree we obtain a good approximate load of [n/p] for each processor. The strategy is to take from our tree the first [n/p] nodes and to allocate them to the first processor the next [n/p] nodes allocate to the second processor and so on and for the last processor we allocate the last [n/p] nodes.

This strategy has it's drawbacks also. Let's think in the case that we have more processors than nodes then the splitting will not be very efficient because we have a lot of communication between processors and less computation per processor. Then the condition for this strategy to work is to have n considerably bigger than p.

Example:



A natural question comes now to our mind: How good is the partitioning introduced by Mitchell in the partitioning strategies that we have seen until now? We had:

rectangle < square < circle

The papers in this field, including [2] where we also have a proof, say to us that actually domain partitions using Peano Space-Filling Curves besides a constant is as good as the circle partitioning.

Thus we obtain:

rectangle < square < Peano SFC < circle

As a conclusion, having domain partitioning, trees as data structure, Peano Space-Filling Curves for unique ordering adding also a splitting strategy we obtain a good recipe for an good parallelisation.

5. Literature

1) Mitchell, W.F., **A Refinement-tree Based Partitioning Method for Dynamic Load Balancing with Adaptively Refined Grids**, (http://math.nist.gov/~WMitchell/papers/reftree.pdf) Journal of Parallel and Distributed Computing,Volume 67 , Issue 4 (April 2007)

2) H.-J. Bungartz, M. Mehl und T. Weinzierl: **A Parallel Adaptive Cartesian PDE Solver Using Space--Filling Curves**. In E. W. Nagel, V. W. Walter und W. Lehner (Hrsg.), *Euro-Par 2006, Parallel Processing, 12th International Euro-Par Conference*, Band 4128 der Reihe LNCS, S. 1064–1074. Springer-Verlag, Berlin Heidelberg, 2006.

3) M. Griebel and G. Zumbusch, **Hash-Storage Techniques for Adaptive Multilevel Solvers and Their Domain Decomposition Parallelization**, Proc. Domain Decomposition Methods 10, Contemporary Mathematics, Vol. 218, (AMS, Providence, 1998) 271-278.

4) M. Mehl: **Ein interdisziplinärer Ansatz zur dreidimensionalen numerischen Simulation von Strömung, Stofftransport und Wachstum in Biofilmsystemen auf der Mikroskala** (http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2001/mehl.pdf) Dissertation, TU-München, 2001.

5) http://www.cs.utexas.edu/users/vbb/misc/sfc/Oindex.html