An algorithm to traverse N-dimensional grids with one loop

Marcelo Lares
Instituto de Astronomía Teórica y Experimental (IATE - UNC/CONICET)
Córdoba, Argentina

Nested loops in astrophysical problems

Many astronomical applications of high performace computing rely on the traversing of high dimensional parameter spaces. The programs writen to perform this tasks have a nested loop at its core. This structure is used, for example, to traverse N-dimensional data grids, to search for a maximum of a discrete likelihood function, to search for neighbors, to compute properties of close object pairs, etc. The piece of code used to perform this task is simple and straightforward, but:

This introduces an increase in source code complexity and multiple nested loop structures, wich impose limitations on the implementation of parallelism. Hereby I propose a novel algorithm to traverse grids of arbitrary dimensions with a single automatically adaptable loop algorithm. This algorithm is based on the binary representation of each step and on algebra modulo the number of dimensions, and introduces minimum overhead, since in general the heavy part of the computations is located inside the loop, and involves great amounts of data. It also allows to obtain the ideal data decomposition for a given number of threads and is trivially scalable, due to the reduction of the nested loops to a single "for ... do" strutrure, reducing starvation for a given granularity. This allows the scalability of the problems in parameter space, without the need of intensive code maintainance.

This is ideally suited to implement problems such as: MonteCarlo Markov Chains in multiparameter spaces, fitting problems, maximum likelihood, neighbor search, properties of pairs (e.g. correlation functions), and others. The code is easily paralelizable using standards suchs as OpenMP or MPI.

Method: Binary representation of base vectors.

A grid is constructed in an N-dimensional space. The number of bins in each dimension may be different, so that the total number of bins is the product of the bin number on each dimension or parameter. The grid is traversed using a set of directions defined so that they connect each cell to all the neighboring cells. For example, in two dimensions the set of directions is: { (1,1), (0,1), (1,0), (1,-1) }.
To go through all the cells in the grid, the algorithm starts from any cell and jumps to any direction until the first cell is visited for second time. Then another jump is performed to the next direction drawn from the set of basic directions (figure 1).

In order to obtain the set of basic directions, the following procedure is implemented:
Write all the natural number lesser than the number of dimensions in base 2. These numbers represent a set of vectors that span the N-dimensional space, and can be thought of the aristas of a given cell of side=1. The all the 2-combinations of vectors are obtained and included to the set of directions.

To travel the grid, starting from any point (figure 2), the next basic direction is used each time a cycle is obtained in the grid.

The core of the algorithm is shown in "code 2". A routine is called from the main loop,
and each time the routine computes teh next cell using the information of the current cell

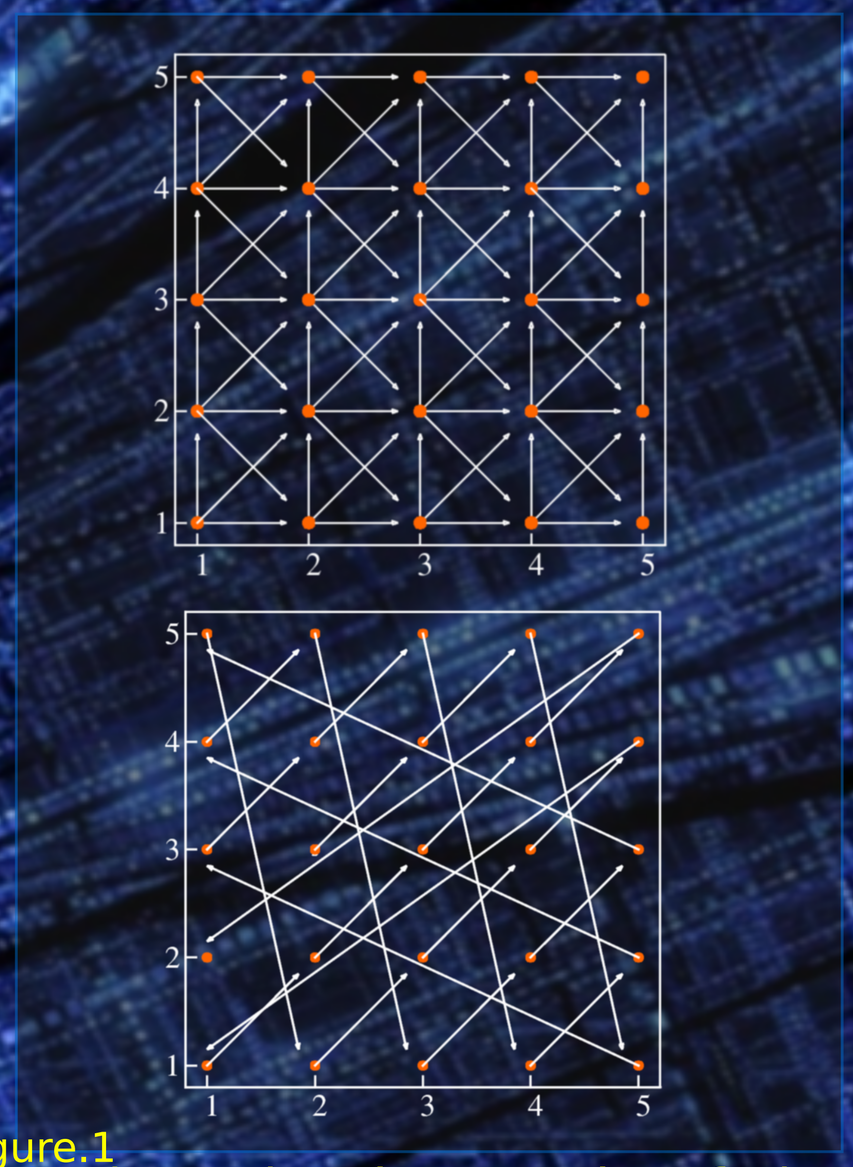

Figure.1
A 2-Dimensional example of the path obtained from the algorithm. Upper figure: pair relations between neighboring cells. The order with wich the algorith go

The code

Number of directions in N-dimensional space
Obtain directions from a binary scheme
Traverse the grid

code.1

```
1. for all b_i ∈ {cells} do
2.         cell ← TraverseGrid
3.         p ← mark(cell)
4.         for all b_j ∈ {neighbors} do
5.                 n ← mark(neighbor)
6.                 if b_i ≠ b_j then
7.                 (Make here pair computations)
```

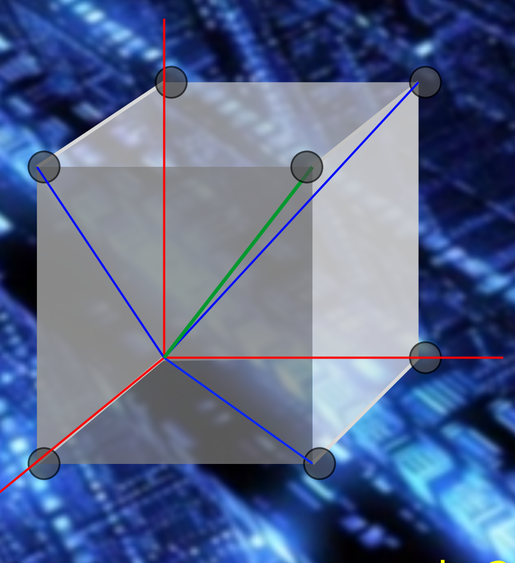

Figure.2
A scheme showing a 3-D cube and all the possible directions: The directions given by

code.2

```
TraverseGrid ( i, Nbins, idxs )

1.  i ← (index of pivot cell on the grid)
2.  Nbins ← (Number of bins in each dimension)
3.  N ← (Number of dimensions=∑(Nbins))
4.  P ← 1
5.  for d ∈ {dimensions} do
6.          P ← P*Nbins(d)
7.          Q ← (i-mod(i,P))/P
8.          idxs(d) ← mod(i,Nbins(d)) + 1
9.          k ← idxs(d) + Q
10. return idxs
```