All the algorithm process can be summarized in the C code described below. The names inside the comments will be used as references to each procedure in the rest of the text.

```c
Marching_Cubes()
{
    Read_Configurations_Table();
    Calculate_Cubes_Configurations(); /* Flags */
    Calculate_Verify_Cubes_Polygons(); /* Marching */
    Calculate_Normals(); /* Normals */
    Generates_PolyTriangleStrip(); /* Triangles */
}
```

Besides the techniques to optimize the algorithm, each procedure of the algorithm was optimized to run in vector machines. The good results of this implementation in terms of performance encouraged us to port it to a parallel machine.

3. **Algorithm Parallelization Methodology**

As mentioned before, the idea was not to deeply modify the algorithm to port it to a parallel machine. The idea is to divide the 3D data grid (Fig. 1) between all the processors of the machine.

The complexity of data division for parallel processing is proportional to the dependencies among data, in order to assure data consistency between processors. Fortunately, in the case of this marching cube implementation, the only dependency between adjacent grid cubes occurs in the normals calculation process.

As described before, the normal of a vertex is calculated considering a weighted average of the polygons normals that share it and a poor grid data division might cause vertices to lose connections with polygons (Fig. 9). The solution is to keep an extra layer of border grid points during data division (Fig. 10).

![Figure 9: Data grid division and the problem of vertices connections.](image-url)
4. The machine and the programming methods

The Connection Machine [HILL93] uses massive parallelization and it was produced by the Thinking Machine Company. This machine has two models: CM-2 and CM-5. The CM-5, the last model, differs of the CM-2 in the ability of the nodes to run instructions in a independent way and in the organization of the input/output mechanism. Also, the CM-5 uses more powerful 64 bit processors.

This kind of machine is structured to allow the execution of a program in a massively parallel environment. As opposed to a conventional vector supercomputer that uses a few powerful and expensive processors, the CM works with a great number of cheap processors (the RISC Sparc) with local memory. The machine architecture does not impose a theoretical limit for the number of processors, but it was projected to support until 16000 processors and the largest machine built has 1024 processors.

The CM allows the creation of processor partitions. A partition is a set of processors controlled by a processor called the controller. The CM-5 at the University of Wisconsin, per example, has two partitions, each with 32 processors. One of these partitions has nodes with vector facilities.

There are two basic models to program in a CM: the MP (message passing) and the DP (data parallel). In the CM guides, these methods are referenced as MIMD (Multiple Instruction, Multiple Data) and SIMD (Single Instruction, Multiple Data), respectively.

For this application, MP programming is a simple job as it is only necessary to define how to divide the data among the nodes and how to execute the same copy of a program in each node. Each node processes a subset of the original data. The controller runs another program that controls the execution of the node programs. After data processing in each node, the results are sent to the controller. At this time, the controller can merge the output data. Another interesting option is to send the results directly to the network. This allows the execution of the node programs in an asynchronous manner but requires the receiving machine to merge output data from the nodes.

MP supports the programming languages C, Fortran and Pascal. The MP programs only call special communication and synchronization routines. Depending of the application, the node program can be the original sequential program with a few modification related to the calling of these communication and synchronization routines.

In the case of the marching cube implementation, the main problems of this method were related to the control of the node synchronization and the definition of the best way to divide the data among the nodes and to merge the output data. The synchronization was, in general, only necessary for communications and it was simple to insert this operation in the program. The data division and output merging were also simple and used a scheme similar to the data parallel implementation.

The data parallel implementation requires the use of a special programming language called C* ("C star") that contains resources to allow the parallelization of the program in a semi-transparent way to the