VECTORIZING SURFACE MODELING ALGORITHMS

David L. Humphrey
Steven Zoraster
ZYCOR, Inc.
2101 South IH-35
Austin, Texas 78741
(512) 441-5697

BIOGRAPHICAL SKETCHES

David L. Humphrey is a Project Manager in ZYCOR's Research Division. Mr. Humphrey received a Master's degree in Mechanical Engineering from the University of Texas in 1978. He has previously worked for Lawrence Livermore Laboratory. At ZYCOR, Mr. Humphrey is responsible for development of gridding and grid processing algorithms and is in charge of developing cartographic applications for array processors.

Steven Zoraster is Manager of ZYCOR's Research Division. Mr. Zoraster received a Master's degree in applied Mathematics from UCLA in 1975. He has previously worked in the field of radar and signal processing. At ZYCOR, Mr. Zoraster has been responsible for research in cartographic automation, software optimization, and advanced algorithm development.

ABSTRACT

Recent reductions in the cost of computer hardware makes the use of "supercomputers" and array processors to execute cartographic algorithms more practical. However, considerable reorganization in algorithm implementation may be required to make efficient use of these architectures. The purpose of this article is to acquaint the reader with the general concepts of vector computers and some of the design issues which arise in adapting cartographic algorithms for execution on these machines. An example involving modelling of a faulted surface is presented.

INTRODUCTION

Though the speed of computer hardware has doubled approximately every three years since the mid-1950's, this trend has become difficult to maintain for standard sequential machine architectures. Recent advances in computing speeds have been made using alternative architectures such as array processors (APs) and "supercomputers" which employ multiple computing elements in parallel operation. These machines will be collectively termed vector processors in this paper.

Until recently, vector machines were so expensive that their use was justified only in special situations for which they were designed such as the solution of partial differential equations encountered in continuum mechanics calculations. However, the ever decreasing price/performance ratios for

these machines, particularly for APs, have made it more feasible to use them in selected cartographic applications. Investigations currently in progress at ZYCOR are aimed at assessing the suitability of the following algorithms for execution on vector hardware:

- o gridding from point data,
- o gridding from contour data,
- o grid refinement
- o grid smoothing, and
- o grid contouring.

The purpose of this paper is to acquaint the reader with the general concepts of vector computers and some of the design issues which arise in adapting cartographic algorithms for execution on these machines. An example involving modelling of a faulted surface is presented.

CONCURRENCY IN SOFTWARE

The successful application of a parallel machine to a real-world problem depends strongly on the amount of parallelism inherent in the algorithm being executed. Hardware designers have analyzed typical programs for various applications to determine baseline requirements for their machines, i.e., software has driven the hardware design. In this section some of the characteristics of these constructs are examined. Some seemingly sequential processes may be reorganized to enhance their parallel nature while others are inherently sequential and do not use parallel hardware facilities efficiently.

Strongly Parallel Processes

Parallel processes are often executed on sequential machines in a DO loop. Consider the process of multiplying each element of an array A by the corresponding element of an array B to produce an array C. In FORTRAN this might be accomplished by

```
DO I=1,N
    C(I) = A(I)*B(I)
ENDDO
```

where N is the number of elements to be produced. This coding is equivalent to

```
C(1) = A(1)*B(1)
C(2) = A(2)*B(2)
     .
     .
     .
C(N) = A(N)*B(N)
```

A cursory inspection of these operations shows that N independent pairs of operands are used to produce N results. A conventional computer will produce C(1), C(2), ..., C(N) sequentially. Since the operands are independent, that is, they are all defined before execution of the loop begins, the calculations could all be done simultaneously on a computer with N multiplier units. It is these kinds of constructs that APs are designed to execute efficiently.

Reorganization To Enhance Parallel Content

Some kernels may be coded in such a manner as to mask their parallel nature. One such example is

```
set J
DO I=1,N
   D = A(I)*B(I,J)
   C(I) = 1.0 + D**2
ENDDO
```

Note that the parallelism of this kernel is destroyed by the calculation of the temporary scalar D on each iteration of the loop. While this is probably a good implementation for a sequential machine (subscript calculation and primary storage are minimized) an alternative implementation which exploits the parallelism is:

```
set J
DO I=1,N
   C(I) = A(I)*B(I,J)
ENDDO
DO I=1,N
   C(I) = C(I)*C(I)
ENDDO
DO I=1,N
   C(I) = C(I) + 1.0
ENDDO
```

This triple loop configuration requires more coding than the original version but is more desirable in a concurrent processing environment. On some computers the length of the machine code necessary for this implementation is shorter than for the former since each loop generates instructions which invoke hardware rather software implementations of loops.

Constructs Which Reduce Parallelism

Some kernels are inherently sequential in nature and usually cannot use concurrent processing hardware efficiently. Sources of interference in a kernel include:

1. A subroutine call
2. An I/O statement
3. Nonlinear indexing of array subscripts
4. Branches to other sections of code
5. Recurrence relations (use of the results of an iteration on subsequent iterations)

Implementations with nonparallel kernels may often be restructured to move the interference source outside the parallel areas. This may involve additional coding and storage but the result may be much higher execution speeds.

AN EXAMPLE FROM FAULTED SURFACE MODELLING

Processing of fault data in filtering of subsurface grid models presents an example of the need for a new programming perspective in using vector processors efficiently.

Biharmonic Grid Filtering

A well accepted goal in gridding random point data is a surface with minimum total curvature consistent with the input data. The minimum curvature criterion leads naturally to the use of a biharmonic smoothing filter (Briggs, 1974). The biharmonic operator is implemented using a 13 point stencil as shown in Figure 1. Normalized weights are applied to each of 12 nodes surrounding the current target node to obtain a new "filtered" value for the target node. The operator may be applied in a number of ways. One method which provides rapid convergence on sequential machines is successive over-relaxation.

Faults in Subsurface Modeling

Gridding of random point data is a standard cartographic problem. In the energy exploration industry the input data and the resulting models usually represent subsurface geological formations. In this domain, geological faults mark discontinuities in the subsurface. They are drawn by geologists or geophysicists on basemaps and hand digitized to provide input to gridding and filtering algorithms. Information should not propagate across a fault during mathematical operations such as grid filtering. In situations where a standard biharmonic operator would extend across a fault a special operator must be developed. Figure 2 shows an example of this situation.

While it is not hard to write software to perform grid filtering in the presence of faults, it is difficult to make such code run efficiently. Since everyday problems can involve grids with more than 500 rows and columns and thousands of fault segments, efficiency is important.

As can be seen in Figure 2, a fault which impacts the filter operator at a grid node will cross one of four search lines radiating from that node. Therefore, in performing biharmonic smoothing in the presence of faults it would seem that the "natural" order of operations is:

```
Loop over grid nodes

    Loop over search directions to gather
    surrounding node values

        Loop over fault segments to detect fault crossings
        of search lines

            Apply appropriate biharmonic operator.
```

While this approach is acceptable on sequential hardware, different methods are required for best performance on array processors or supercomputers.

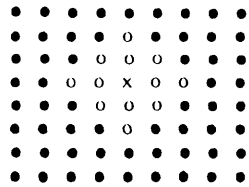

Figure 1(a). Biharmonic Operation Stencil. Point to be filtered is at x. Values at points marked by o are used in filtering x

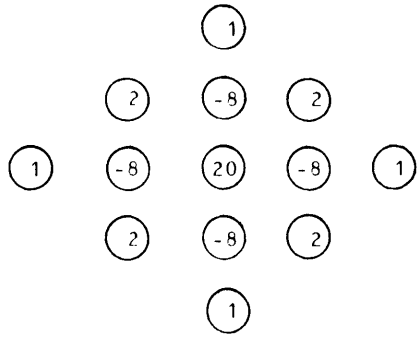

Figure 1(b). Weights of Points in the Biharmonic Operator Stencil

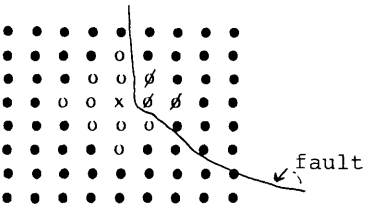

Figure 2. Biharmonic Operation Stencil. Point to be filtered is at x. Values at points marked by o are used in filtering x. Values at points marked by ϕ are used in standard operation but are blocked by fault in this case.

The Solution for Vector Hardware

A difficulty with the algorithm described in the previous section is that it requires looping over many fault segments, no more than one of which is likely to intersect a search line at a critical point. This requires use of conditional logic, something which should be avoided in vector software.

The problem description along with the programming guidelines provided in Section 2 provides hints on how to develop better vector code. First, the fault data tends to occur in short segments. If all segments were short enough it would be possible to guarantee that each individual segment crossed no more than one search line. Second, search directions for nodes on the same grid row or column or the same 45 degree profile overlap. Therefore, initial intersections finding can be organized on the basis of shared search lines rather than on individual grid nodes. Finally, loop reorganization is often an effective "trick" for optimizing performance on vector hardware. In this case, the best loop reorganization involves processing the fault data first rather than almost last. Combining these ideas leads to the following approach which is more suitable for vector processing. Three software modules are used. The first is:

```
Normalize the fault data to the grid coordinates

    Loop over fault segments to detect and mark fault
    crossings of search lines

        Sort and store fault crossing for each search
        direction
```

The second module is:

```
Loop over grid nodes

    Loop over search directions with fault crossings
    to develop special biharmonic operators for each
    grid node position.
```

The third module is:

Loop over grid nodes

Loop over search directions to gather surrounding node values

Apply appropriate biharmonic operator.

For each node, values of zero are assigned to biharmonic filter weights which correspond to data points across faults. This assignment is completed in the second module so that the third module requires no direct information about the fault structure beyond what is encoded in the filter operator for each node. Many filter operators are developed, but a simple data structure allows them to be shared by different grid nodes.

In the first module fault data is processed to guarantee that each segment is shorter than the separation between adjacent grid rows and columns. Longer segments are subdivided. Fault data is also normalized to the minimum grid X coordinate and intercolumn spacing so that the coordinates of each column in the grid and the corresponding vertical search profile are numbered 1 to N. Combining short segments and this normalization mean it is possible to generate significant information by performing the following operation for each fault segment ((xi,yi),(xi+1,yi+1)).

Ki = ABS (INT(xi+1)-INT(xi)) * MAX (INT(xi+1),INT(xi)).

Here ABS, INT, and MAX are respectively the absolute value function, the integer truncation function, and the maximum value function. A little thought should convince the reader that Ki is 0 if the segment does not cross a vertical search profile (because INT(xi) = INT(xi+1)) and is the number of the vertical search profile crossed otherwise.

The ABS, INT, and MAX functions are implemented as vector operators in nearly all supercomputer and array processor subroutine libraries. After the calculation of the Ki, it is possible to continue with calculation of search line intersection points by simple, vectorizable, algebraic operations. These intersections can be sorted and stored for latter retrieval. Equivalent operations can be performed on horizontal and 45 degree offset search profiles.

OBSERVATIONS AND CONCLUSIONS

Though the investigations are still in progress, several observations may be recorded at this time.

To take full advantage of these specialized hardware architectures, major organizational changes are usually required to the software. These changes are usually associated with processing information in large groups or with avoiding conditional processing.

Algorithms organized for vector computers usually require

significantly more memory for the storage of intermediate results such as in the preceding example. As such, only machines which support a large memory address space are should be considered for cartographic applications such as grid filtering and contouring. Hardware-software configurations which include sustained data transfers to a disk drive or host computer will probably leave the computational units idle a large fraction of the time.

Implementation of an algorithm for vector processing requires that the implementation be tailored for the hardware rather than the programmer. As such, such programs are usually harder to comprehend which imposes an extra burden on the maintenance programmer. However, once the programmer has shifted his thought processes to vector mode and the algorithm has been reformulated, transportation of the code to other vector machines should be straightforward.

A final observation is that the 64-bit accuracy offered by some vector computers is not necessary for most cartographic computations such as grid initialization, filtering, and contouring.

REFERENCES

Briggs, I.C., "Machine Contouring Using Minimum Curvature", Geophysics, Vol 39., No. 1, Pages 39-48, February 1974.