Distributed Force-Based Thinning And A General Distribution Method

D. Molaro
C. Jennings
J.R. Parker
Department of Computer Science,
University of Calgary,
Calgary, Alberta, Canada T2N-1N4

ABSTRACT: A recently developed thinning algorithm, based on computing repulsive 'forces' acting on each object pixel, produces nice skeletons, but involves some very intensive computations. As a result, the method takes a long time to thin any real image, when compared against other existing methods. It can be made practical by distributing the computation across a network of workstations. This has applications to other computationally difficult image processing and vision algorithms, and has been generalized and made relatively simple to do.

I. Introduction

Force based thinning [5] is a recently developed technique for generating very good skeletons of digital images. This thinning algorithm also requires a large amount of computation for each pixel of the image. Further use and acceptance of the algorithm was being hindered by the too-long turn around time, and it became important to speed up the calculations somehow. Force based thining, like many image processing algorithms, relys heavily on local information, making it an excellent candidate for distributed processing. After extensive analysis, the idea of distributing the algorithm across a number of computers was the solution to computing reasonable images in reasonable time. After months of design, implementation, and testing, we now have a general structure for distributing similar algorithms and have accumulated valuable experience in practical distributed computation.

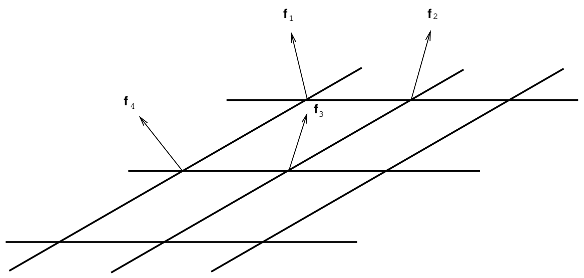
The complexity of many problems is a significant barrier to their being solved in a reasonable length of time. Presently these problems are solved through the application of algorithm design, data structure optimization and code tuning. If all these techniques fail to produce a practical running time the last solution is to use a faster computer. The most complex problems have required the use of the most expensive of computational resources, the super computer (for the purposes of this discussion, a super computer is a single or multiple processor computer with a performance range of 100 to 1000 MIPS (million instructions per second). Fortunately, many of the computationally complex problems for which super computers are used can be effectively parallelized (broken down so that different processes work in parallel on different machines).

II. Distributed Processing: Thinning As A Case Study

As the number of processors participating in the computation increases the time to compute a given problem should decrease proportionally; this would certainly be the case with a process that has zero-communications overhead. However, it is plain that no real algorithm will be in this class. Force based thinning is close to the optimal algorithm for this, since a very small amount of data is transmitted and received for each operation, which themselves are slow. An almost linear speed up as a function of the number of processors would be expected, with saturation of the network at a large (>30) number. A brief description of the thinning method is is order, before the distribution method and results could be reasonably interpreted.

The algorithm first locates the background pixels having at least one object pixel as a neighbor and marks them. These will be assumed to exert a repulsive 'force' on all object pixels: the nearer the object pixel is to the boundary the greater is the force acting on it. This force field is mapped by subdividing the region into small squares and determining the force acting on the vertices of the squares. *The skeleton lies within those squares where the forces acting on the corners act in opposite directions.* Those squares containing skeletal areas are further subdivided, and the location of the skeletal area is recursively refined as far as necessary or possible. The change in the direction of the force is found by computing the dot product of each pair of force vectors on corners of the square regions:

$$d_1 = \vec{f_1} \cdot \vec{f_2} \qquad d_2 = \vec{f_2} \cdot \vec{f_3} \qquad d_3 = \vec{f_1} \cdot \vec{f_4}$$



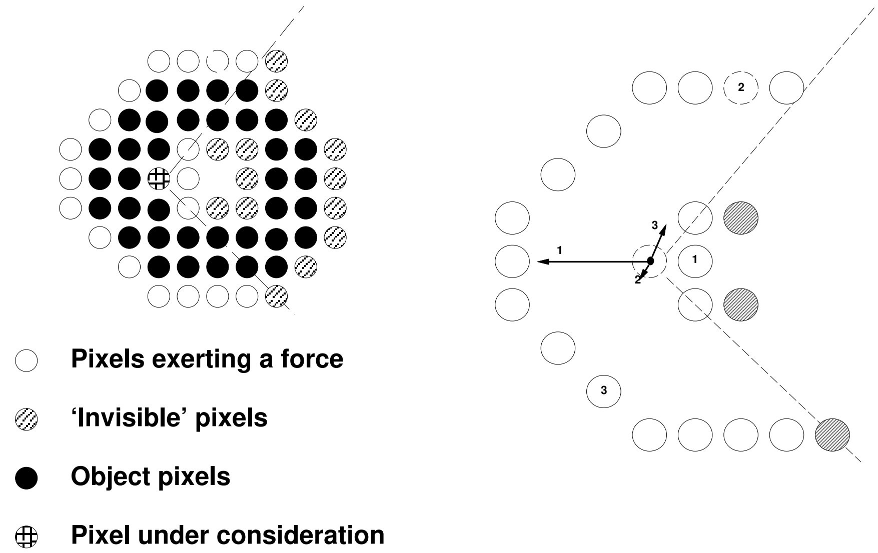
If any one of d_1, d_2 or d_3 is negative then the region involved contains some skeletal area.

To compute the force vector at each pixel location is time consuming. For each object pixel a straight line is drawn to all marked pixels on the object outline. Lines passing through the background are discarded, and for each of the remaining lines a vector with length $1/r^2$ and direction from the outline pixel to the object pixel is added to the force vector at that pixel.

This is done for all object pixels. Then recursive subdivision can be used to refine the positions of the skeletal areas. From any end points of the skeleton found in the previous stage, we consider growing this skeletal line until it hits another skeleton or an edge. If it hits another skeleton then we consider the grown area to be part of the skeleton, otherwise if it hits an edge we ignore the grown section.

The skeleton is grown by looking at the three neighbor pixels to the current end point that are in approximately the direction of the skeleton line. The neighbor pixel with the smallest force magnitude is chosen as the pixel to grow to. This growing is iterated until the growth front hits an edge or other part of the skeleton.

III. Distributed Objects: A Simple System For Image Processing



(a) (b)

Figure 1 - Computing the force at an object pixel. (a) Only 'visible' pixels will exert a force on a target. (b) The vector sum of the forces is computed.

It was desirable to produce a system that could be widely and easily used, so it was decided to use hardware and software facilities that were both well known and portable. As with any implementations there are tradeoffs in selecting a target system. This revision of the program is to be implemented using TCP/IP STREAM sockets in the UNIX environment,with C++. These selections were made for the following reasons.

- The UNIX system was chosen due to its availably and network interface. With the standardization of the UNIX system among several vendors, portability is ensured [6].

- TCP/IP was chosen as a network interface due to its availability and general acceptance [3]. While UDP/IP would provide a slightly more efficient transmission the programming required for getting a reliable transmission would require too much effort at this point in the project. Many non-UNIX system are supporting the socket abstraction for a network interface. Additionally, the system could be ported to the OSI network model [8] with little effort.

- We use object-oriented design, which means an object oriented language; Network programming requires in a systems programming language such as C. C++ is therefore the logical choice [7]: it provides object oriented facilities and a C like interface to systems level routines.

The distribution model has been implemented as as C++ class that is to be inherited. The derived class then has the attributes that it can then be transmitted and received by all who use that class. It has been defined as:

```
class Connection
{
 public:
   enum ConnectionType  { Client,Server };
 private:
   ConnectionType memType;
   int s;
   char *server;
   char *service;
   virtual void *encode(int *length);
   virtual Boolean decode(void *data,
                      int length);
 public:
   Connection(char *pserver,char *pservice);
   Connection(char *pservice);
   ~Connection();
   Boolean pending();
   Boolean receive();
   Boolean transmit();
};
```

The software can be described in terms of *methods,* which are simply operations of the data within the class. A user wanting to make an program work in parallel would design a class that encapsulates the expensive computations. The *public methods*, or those that are accessible to the programmer, are described as:

Connection(char *pserver,char *pservice);

This constructor attempts to attach it self to a server node on the specified node advertising under the specified service name. If there is no currently advertising server specified by this pair the constructor fails and will cause the program to exit.

Connection(char *pservice);

This constructor advertises the services of this node as a server for the derived class. The method does not return until a client node has attached.

˜Connection();

The destructor closes down the connection between client and server, the destructor will not return until all of the data on the communications channel has been drained.

Boolean pending();

The pending method returns True if there is any data waiting to be read off the communications channel.

Boolean receive();

The receive method reads any pending data off the communications channel and decodes it into the class.

Boolean transmit();

The transmit method encodes the data and sends it to the remote node.

In addition there are 2 so-called *private methods:* those that, or those that are used internally by the class. These methods are to be defined as virtual functions by the user.

void *encode(int *length);

The user should define the encode method to take all of the data that is to be transmitted to a remote node. This method is called by the transmit method. If the encoding fails for any reason the method should return a NULL pointer. The memory returned should be dynamically allocated as the transmit method will delete the memory once it has been transmitted.

Boolean decode(void *data, int length);

The user should also define the decode method, which takes data transmitted to this object and decodes it into the structures recognized by the object. Typically this method will simply undo what ever the encode method does. If for some reason the decoding method fails the value of False should be returned, True otherwise.

This class encapsulates all of the neccessary network code so that a user does not have to be a systems programmer to write network programs.

III. Implementation And Results

Of the options available when implementing force based thinning on a parallel machine, the most natural solution is to impose a ‘‘bag’’ topology on a set of nodes to act as a processing resource and construct a task manager node which dispatches jobs to processors and accepts their results. This is illustrated in Figure 2; in this case the object that is to be distributed is the force processor. A pixel element is accepted by the processor and the force for that pixel is computed. The definition for the force processor object is:

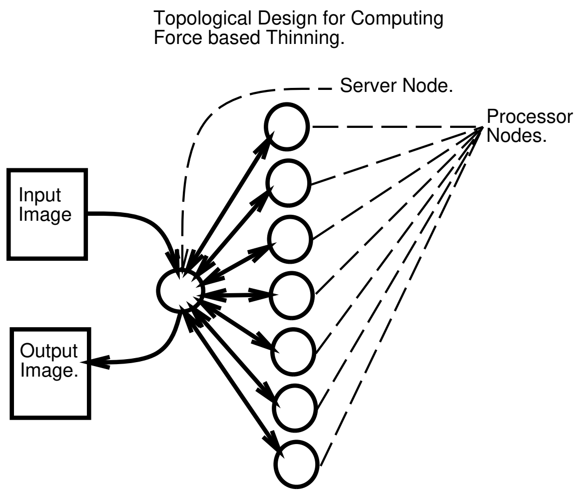


Figure 2 - Structure of the distributed thinning system

```
class Processor : public Connection
{
   Point input;
   Force output;
private:
   virtual void   *encode(int *length);
   virtual Boolean decode(void *data,int length);
public:
   int   atprocessor;
   Boolean Busy;
   Processor(char *server,char *service);
   Processor(char *service);
   Point getPoint();
   void  setPoint(Point p);
   void  setForce(Force f);
   Force getForce();
};
```

The force based thinning algorithm was run on a sample image using a varying number of processors. The processors were of all the same type, in this case Sun Sparc Station 2's and 1's, Running Sun OS 4.1.2. The communications between processing nodes is Ethernet. Experiments were run to compare the performance of the original implementation with the distributed implementation. Those results are summarized as:

Number of Processors	Time to Compute
1	903 sec
1	1976 sec
1	897 sec
2	477 sec
3	302 sec
4	226 sec
5	187 sec
10	125 sec

and appear graphically in Figure 3. The first result is that of the original implementation, and can be used as a basis for comparison. The second result is executing the distributed implementation on a single cpu; that single cpu handled both the client and server side of one processor. The third result is using a single distributed processor disjoint from the server node. All results beyond that were accumulated the this manner.

IV. Conclusions

It can be seen from these results that a near linear speed up in processing time can be achieved

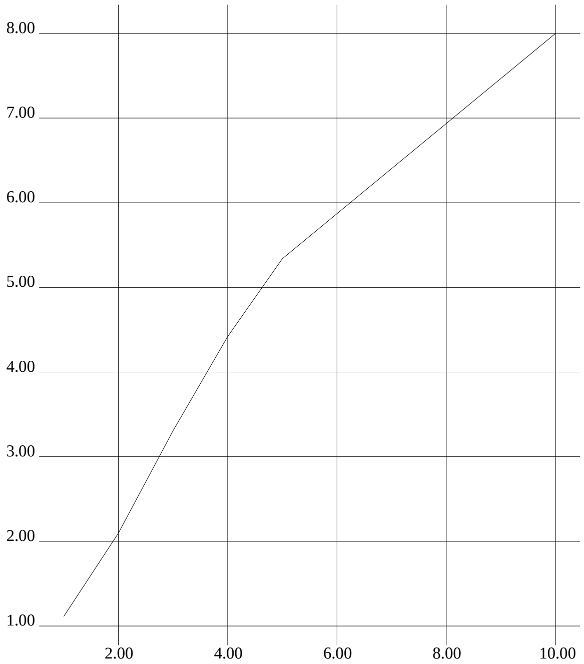


Figure 3 - Results of the force based thinning method distributed over a network.

over a small number of processors and performance tails off as the communications channel becomes saturated.

This work is significant to the research community because it gives the average researcher the ability to do work into distributed and parallel computation, without having a special purpose machine. This work is significant to the commercial community because it can use available technology to compute very hard problems in parallel on economically priced hardware.

The media (ethernet), although rated for 10 Mega-baud, only gets 30% of sustained throughput. This means that the calculations performed on each workstation should be significant, or the overall computation becomes I/O bound. The ideal situation is a set of difficult problems on relatively little data, such as the Fourier transform, restoration filters, and similar problems.

The overall result is that an easy to use package has been produced for allowing the distribution of hard problems across a network of computers. This system is being used at the Univeristy of Calgary for vision and image processing problems with great success, and it is anticipated that, with an increase in the network bandwidth, the computational power of our net would compare favorably with that of a small supercomputer. Further work and measurements are proceeding with that as the goal.

V. References

1. Goscinski, A., **Distributed Operating Systems, The Logical Design,** *Addison-Wesley,* New York.
2. Hoare, C.A.R., **Communicating Sequential Processes,** in *Concurrent Programming,* (A.D. McGettrick, ed.), Addison-Wesley, 1988.
3. SUN Ltd, **SUN Network Programming Guide,** Sun Microsystems Ltd, Mountain View, CA.
4. Parker, J.R., and Ingoldsby, T R., **Design and Implementation of a Multiprocessor for Image Processing,** *Journal of Parallel and Distributed Computing,* 9, 1990. Pp. 297-303.
5. Parker, J.R. and Jennings, C., **Defining The Digital Skeleton,** *SPIE Vision Geometry Conf*, #2522, Boston, MA, November 1992.
6. Stevens, W.R., **Advanced Programming in the UNIX Environment,** *Addison-Wesley,* New York. 1992.
7. Stroustrup, B., **The C++ Programming Language,** *Addison-Wesley,* New York.
8. Tanenbaum, A., **Computer Networks,** *Prentice Hall Inc,* Englewood Cliffs, NJ.