EPCC-SS99-02

Parallel Rasterisation for GIS

Thomas Ashby

Abstract

Geographic Information Systems are growing in their need for computing power. The manipulation of very large volatile datasets with increasingly complex operations is an obvious choice for high performance parallel computing. This report looks at a parallel implementation of vector to raster conversion for BSi NTF level 4 records.

1 Introduction

This report is based on work done for the Edinburgh Parallel Computing Centre accomplished over a ten week period that started on the 12th of July, 1999. The initial one and a half weeks were spent on training courses concerned with the theory and application of parallel computing, and the remainder of the time was spent on the project for the parallel conversion of BSi NTF level four vector information to a raster form for a GIS.

The background section gives some information regarding the role and development of Geographical Information Systems (GIS), and why high performance parallel computing is needed to meet the demands of modern GIS applications. It also details the history of the wider project, to which my SSP project is a fairly minor addition.

The project description section gives an account of how the work done and problems encountered during the course of the project. It is broken up into further relevant subsections.

The timings section documents various test conducted on EPCC machines and the results obtained

The conclusion discusses the results from the timing section and looks at the work left to do.

2 Background

2.1 GIS's and their role

Crudely speaking, GIS's are electronic maps - a marriage of Computer Assisted Cartography and a database. However, unlike traditional maps, data storage and data presentation are inherently separate in a GIS. The upshot of this is that GIS's are more flexible than maps, which are static and tend to compromise to be of use in a wide range of purposes. A more thorough definition gleaned from [1] defines them as computer systems for the:

- acquisition and verification
- compilation
- storage
- updating and changing
- management and exchange
- manipulation
- retrieval and presentation
- analysis and combination

of geographic data. Geographic data is defined as "information on the qualities of and the relationships between objects which are uniquely georeferenced".

The roles of GIS's are many fold. For example, spatial analysis and associated information on traffic flow could be used to calculate the shortest or quickest driving route between two destinations. Information from disparate databases can be combined to assist decision making, such as balancing the need for conservation against the need for extra housing development. In emergency situations such as potential widespread flooding, calculations could be made using a GIS to identify the most pertinent areas to evacuate first.

2.2 GIS's and parallel computing

The field of GIS's is beginning to realize its potential with an increasing level of interest from across the board of academic and commercial application. Unfortunately, just at this crucial point the endeavour is in danger of being sunk by its own success. The amount of data being made available by GIS's and remote satellite sensing is such that it can barely be stored, let alone subjected to any sort of useful analysis. Although recent leaps in computer performance have been impressive, the increase in power is no match for the *requirement* for power, especially in the more exploratory, interactive or combinatorially explosive kinds of analytical operation.

On top of this, the prospect of really widespread commercial applications such as services provided over the Internet and tourist information systems, and the handling of very large and highly dynamic datasets, is approaching rapidly. Current demand for fairly simple world wide web search queries has resulted in the need for very powerful multiprocessor search engines - handling complex GIS requests at this sort of rate would require massive computing power. Real-time GIS's would require sub-second response times to cope with the volatility of their data, yet another reason to seek a solution to the obstacle of very high volume data throughput.

This is where parallel computing enters the fray. The promise of general purpose parallel computing to handle computationally intensive operations on large amounts of data is potentially the solution to the dearth of computing horsepower currently plaguing GIS's. Until recently the cost of redesigning software to run on parallel machines and their tendency to concentrate on managing computationally intensive jobs rather than data throughput has deterred the GIS community from wholeheartedly embracing the technology, but this is now changing.

2.3 The project so far

2.3.1 History

The program has its roots in a large research project conducted at the University of Edinburgh and funded by the Department of Trade and Industry and SERC, along with several industrial partners.
It is currently being conducted as a collaboration between the GIS Parallel Architectures Laboratory within the Department of Geography at Edinburgh University, and EPCC.
The SSP project is an extension to the project proper, and uses various modules from it as a foundation.

2.3.2 Structure

There are three main tasks to be handled by the program:

- Raster to vector (R2V) conversion
- Vector polygon overlay

- Vector to raster (V2R) conversion

The code is modularised to facilitate re-use (see fig 1).

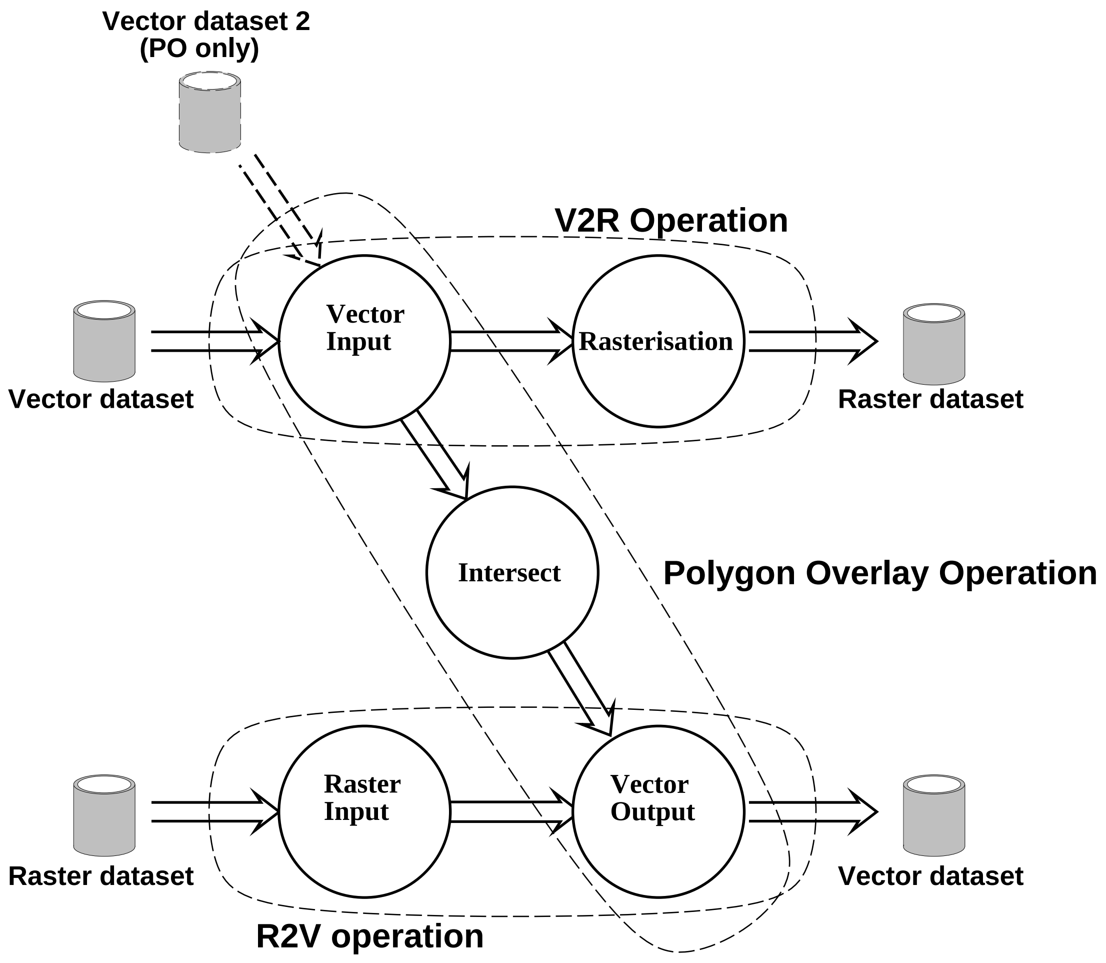

Figure 1: Modular decomposition of code

As can be seen, vector to raster conversion utilises the vector input, rasterisation and raster output modules.

2.3.3 Vector input module

The vector input module takes as input BSi NTF level 4 data sets. It outputs sets of geometry records with their associated left and right attribute values in groups containing all the descriptors present in a horizontal strip of varying height, sorted by their uppermost y co-ordinate (see fig 2). It executes this process in three stages, *Sort*, *Join* and *Geom Attribute Distribution (GAD)*.

SORT During both the Sort and Join phases, there is one source process and two or more pool processes. The source process co-ordinates the actions of the pool processes, and, during

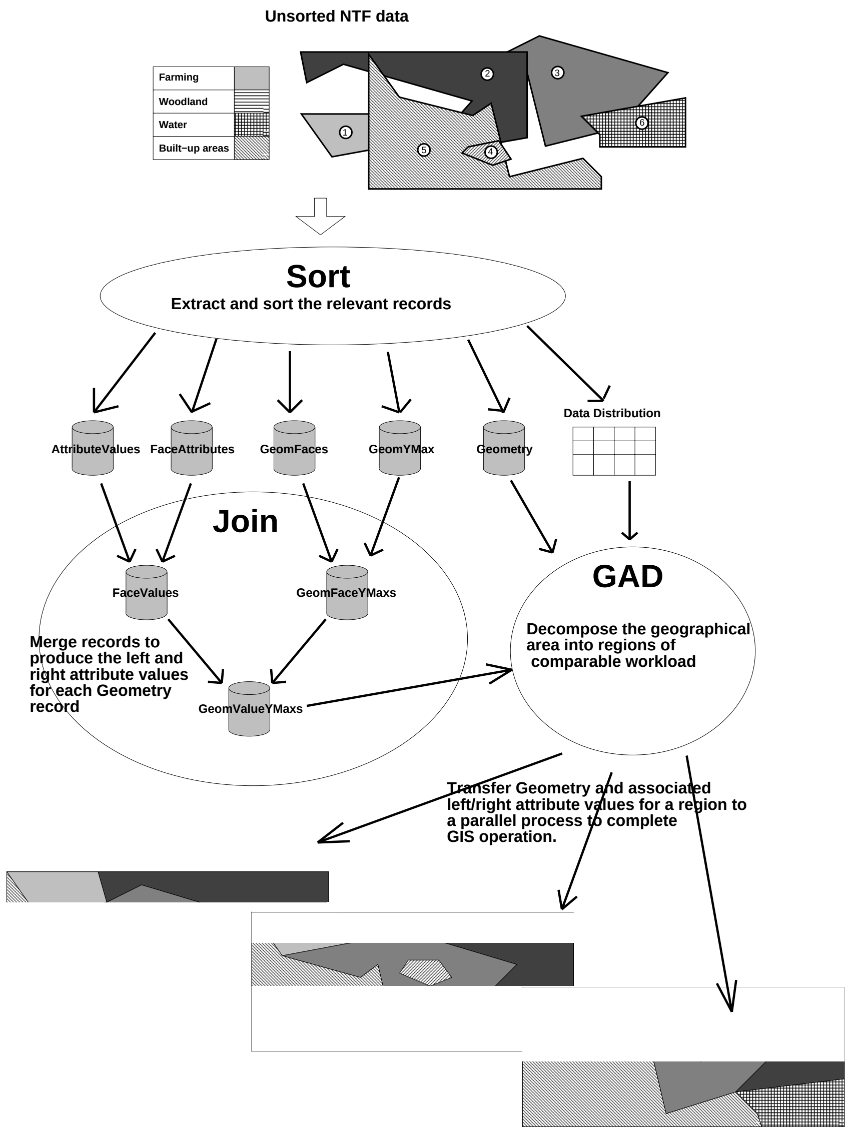
Unsorted NTF data
Farming
Woodland
Water
Built−up areas
1
2
3
4
5
6
Sort
Extract and sort the relevant records
AttributeValues
FaceAttributes
GeomFaces
GeomYMax
Geometry
Data Distribution
Join
FaceValues
GeomFaceYMaxs
GeomValueYMaxs
Merge records to produce the left and right attribute values for each Geometry record
GAD
Decompose the geographical area into regions of comparable workload
Transfer Geometry and associated left/right attribute values for a region to a parallel process to complete GIS operation.

Figure 2: Graphical depiction of Sort, Join and GAD

the Sort phase, gathers information on the distribution of the data over the space of interest. The output from the Sort phase is various temporary files containing the processed and sorted contents of the necessary records.

JOIN During the Join phase, the majority of the output from the Sort phase is merged in two stages to produce GeomValueYMax records. These records contain geom Id's with their left and right attribute values, and y max and y min values, sorted by uppermost y coordinate (**GeomValueYMax** records).

GAD At the start of the GAD phase, the pool processes split into two groups, each containing one or more processes (see fig 3). Processes in the first group are called Geom Attribute Servers (GAServers), and those in the second are called workers. Control of worker processes is then given over to an outside operation (i.e. vector polygon overlay or rasterisation), which can call elements of GAD worker functionality to interact with the GAServers and the source process. This interaction consists of sending a message to the source process to indicate how much work a worker can accommodate (restricted by memory considerations etc.). The source process then replies by issuing a **strip descriptor** message, detailing the vertical extents of the horizontal strip to be assigned to the worker. The source process constructs the strip by starting where the previous one finished, and incrementing the y range downwards by the size of a **mini partition** iteratively until the range contains a close as possible to the maximum amount of work permissible for the worker.

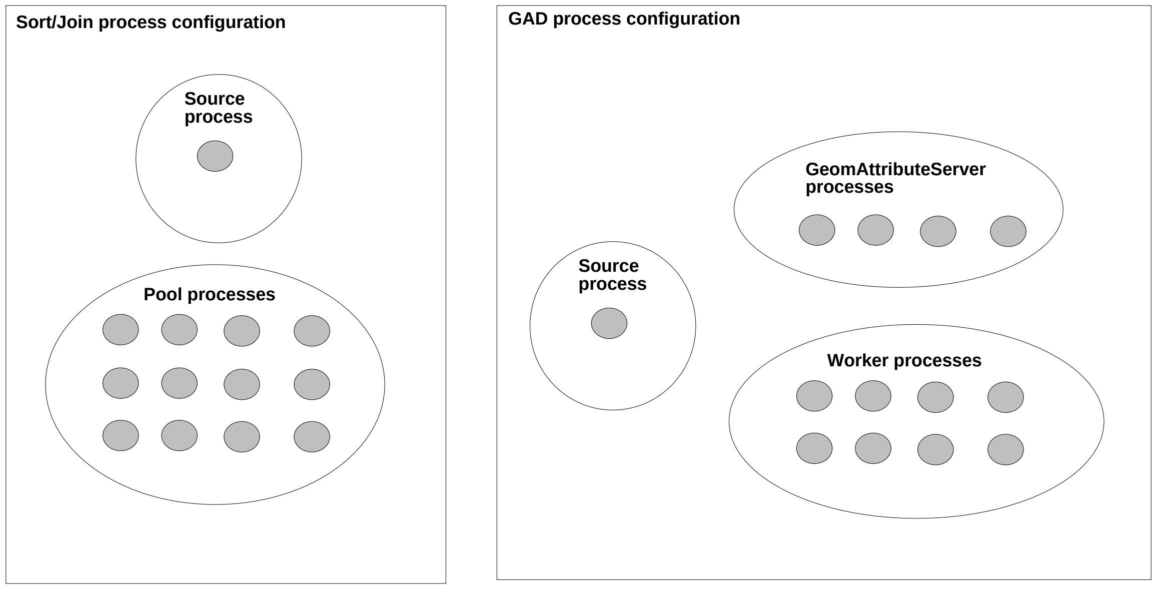
Sort/Join process configuration
Source process
Pool processes
GAD process configuration
GeomAttributeServer processes
Source process
Worker processes

Figure 3: Splitting of processes

When the GAServers receive the strip descriptor, they send any geoms and geom values they own in that range to the worker in question, after prompting the worker with a message telling it how many of each type to expect. GAD worker functionality executed on the worker process then does the final sort and join to enable it hand geoms joined with their left/right attribute values sorted by their maximum y co-ordinate to the parallel GIS operation in charge of the worker. The information is passed by calls to a function that fetches **geom descriptor** structures one at a time in order.

3 Project Description

3.1 Rasterisation Methods

Methods of rasterising vector data exist essentially in a continuum from frame buffer to scan-line algorithms. The continuum starts with frame buffer algorithms where the whole area is held in memory at once. It then proceeds on a sliding scale of dividing the area into smaller strips and rendering each strip separately with a frame buffer style algorithm until the area is a single raster line, which is the scan line algorithm.

Frame buffer algorithms For this method the whole raster image must be held in memory at once. Vectors are drawn according to any standard method, and can then be discarded. Once all the vectors have been drawn, either a seed fill or parity fill algorithm can be used to fill in the resulting polygons. As well as requiring the whole image to be held in memory, frame buffer algorithms are less accurate as they discard the vectors immediately after drawing, and so leave any given pixel with the value of the last vector that intersected it, in the case of a disputed pixel.

Scan line algorithms Scan line algorithms require the input to be sorted by the uppermost y co-ordinate. Before rasterising a given horizontal line, an active edge list is set up denoting edges to be rendered. The active edge list is updated before moving on to the next line by removing edges that have terminated and adding new edges etc. The input is sorted to ease the adding of new edges. The head of the input edge list is interrogated, and if it starts below the current raster line, all the following edges are guaranteed to start below the line as well (assuming we are rasterising an area from top to bottom). Contested pixels can be more accurately rendered with scan line algorithms, as it is less effort for the algorithm to have all the edges present in a pixel before it has to render it, due to memory considerations. It can then use one of a number of decision procedures to resolve the conflict.

As the input to the worker stage is already sorted by maximum y coordinate, and the datasets are expected to be very large, the design specified implementing a scan line algorithm.

3.2 Rasterisation Worker

An overview of the worker function hierarchy is given below:

- RasteriseWorker
  - Rasterise
    - RasteriseStrip
      - RasteriseMethodWrapper
      - UpdateSegmentList
        - GatherContinuations
        - BuildSegment
        - InsertInASL
        - RemoveFromASL
        - CompareSegments (with *qsort*)
      - RasteriseLine
        - CalcPixAreas
          - CompareClipSegments (with *qsort*)
          - CompareIntersections (with *qsort*)
          - SimulateSegments

3.2.1 RasteriseWorker

This function is responsible for transmitting to the source a message detailing how much work it can take on, and receiving the strip descriptor from the source and the relevant geom records and geom value records. The reception of the records takes place in two stages. Initially a size message is received from each GAServer, then a message of the relevant size is received. The function then checks to see that the correct number of records have been received for the strip, and passes the records to Rasterise.

3.2.2 Rasterise

Rasterise calls various pieces of GAD worker functionality to turn the messages into parcels, from which geom descriptors can be retrieved.

3.2.3 RasteriseStrip

Once the geom descriptors have been prepared, the rasterisation of the strip can begin. RasteriseStrip prepares the structure for the active segment list and gets the first geom descriptor - this is necessary in case there is any blank space at the top of the strip to be rasterised. After calculating how high in cells the strip is, the function calls UpdateSegmentList followed by RasteriseLine for each line in the strip. To finish, the output atom must be flushed and closed, although file handling has yet to be finalised.

3.2.4 RasteriseMethodWrapper

This function is local to the rasterise.c file and allows the worker to pick up the rasterisation method stored by the source process. This should really be put into the V2R args in the application args, as I am not certain that this call will work when run on non-shared memory machines (i.e. when rasterise source and rasterise pool are on totally different private memory blocks).

3.2.5 UpdateSegmentList

UpdateSegmentList updates the active segment list (ASL), a structure with pointers to the head and the tail of a doubly linked list of segments. Two pointers are used to enable the merge sort of the new segments from new geom descriptors to be conducted in the manner proposed in the original pseudo-code. This could probably be changed to a single pointer list, and it may even be advisable to do so as merge sorting from the left of the list or some random point in the middle may be more efficient.
The function completes the following jobs in order:

- Removes ’dead’ segments from the list (i.e. segments that are no longer active on the current raster line).
- Checks for and constructs recursively continuations of any segments that finish on this raster line.

- If the segment finishes and has no continuations, call GAD functionality to destroy the geom descriptor.
- Updates the x-extents for each segment in the ASL.
- Updates the x-extents for each segment that was new last time, as this will be a special case.
- Checks the position in the ASL of each segment that was new last time, as they may need to be moved (see [4]).
- Checks to see whether the next geom descriptor becomes active on this line, and constructs its first segment if it does. Checks for continuations if the first segment stops on this line.
- Quicksorts new segments from new geom descriptors.
- Mergesorts all new segments with the ASL.

The new segment table (NST) is contiguous in memory to allow the portion of it that gets filled with new segments from new geoms to be quick-sorted with a call to the library routine ’qsort’, and to ease the operations on segments that were new last time.
The calculation of x-extents should probably be moved out into an individual function to aid the clarity of the code. Having only event point data and the original geoms was looked at as a redesign option, but the amount of extra sorting that would be involved meant that any memory conserving benefits would be outweighed.

3.2.6 InsertInASL and RemoveFromASL

This pair of simple functions deals with the pointer stitching required to add and remove segments from the ASL. InsertInASL can take a number of segments contiguous in memory to avoid the overhead of repeated function calls when doing the merge sort. It uses the ’parent’ member of a segment as the starting place of the search for the insertion point.

3.2.7 BuildSegment

BuildSegment extracts the information from a geom descriptor to initialise a new segment in the NST. It also hands the calls to realloc should the NST table array be too small/large.

3.2.8 CompareSegments

This function is called by qsort to enable the sorting of the new segments from new geoms. It stands currently as it did in the original design, but this doesn’t take into account the vertical positions of the segments if they have identical left x-extents, and so may need changing.

The code for the functions above is detailed in the appendix. Unfortunately the following functions were not fleshed out, as more time was spent debugging the completed functions above than was expected.

3.2.9 RasteriseLine

RasteriseLine uses the ASL constructed by UpdateSegmentList to decide on the attribute values for each pixel in a given raster line. Pixels are rasterised with one of 5 methods:

Arbitrary The pixels take the attribute value of the face to the west of the first segment, and any subsequent segments that occur in the range of the segment are discarded. Transitions after the segment finishes (and no others are active) are filled with the east value until a new segment becomes active.

Priority Pixels are rendered with the highest value attribute present in them. Ranges can be filled until a new segment becomes active or the current highest priority segment becomes inactive. Transitions are filled similarly to the Arbitrary method.

Area Methods For the following methods, each pixel (that is not part of a transition) is considered individually by calling 'CalculatePixAreas' to get the area of each attribute polygon in a contested pixel.

- **Attribute Area**: The pixel is rendered with the attribute that has the greatest summed area of the polygons present.
- **Weighted Attribute Area**: Similar to Attribute Area, except that the area of an attribute is multiplied by its weight, supplied by the second value in the Geom2ValuesYMax records.
- **Polygon Area**: With this method, a pixel is rendered with the value of the largest single polygon present.

3.2.10 CalculatePixAreas, SimulateSegments, CompareClipSegments and CompareIntersections

CalculatePixAreas is called by RasteriseLine when using one of the area methods for disputed pixels. It calls the remaining three functions for various sorting and calculation purposes.

4 Timings

In addition to implementing some of the rasterisation functionality, a number of timing tests for the Sort, Join and GAD stages were conducted. Although Sort and Join had been tested before, they had not been tested on the resources mentioned, and so it was deemed worthwhile to obtain some results.

The first and most comprehensive batch of tests involved timing the program through the Sort, Join and GAD phases, up to where the workers receive all data contained in all the strips. The program was tested with a number of processes on various different machines. The dataset used was a small example made up of 60 mini-partitions, which contained a face with a hole in (i.e. two polygons), and was approximately 1287 bytes in size. The second test involved running the program through the same phases on the same dataset as the first, but also timing the action of UpdateSegmentList on the first strip descriptor as well (unfortunately testing further strip descriptors was not possible - this explains the very quick timings for the results).

It was initially hoped to be able to use larger datasets to investigate the scalability of the code, but this proved unworkable in the time given.

Machines Below is a brief description of the resources used to run the tests:

- **Amber** Sun Ultra 10, with 1 Gb of memory and a 300MHz ULTRASparc processor
- **Lomond front end** Sun Enterprise 3000, with 1 Gb shared memory and 4 x 250 MHz ULTRASparc processors
- **Lomond back end** Sun Enterprise 3500, with 8 Gb shared memory and 8 x 400 MHz ULTRASparcII processors (see *http://www.epcc.ed.ac.uk/sun*)
- **Work Station cluster** Six Sun IPX Work Stations, run as a parallel machine across the network (see *http://www.epcc.ed.ac.uk/sun*)

4.1 Results

Machine: Amber

Number of Processes	Sort Time (seconds)	Join time (seconds)	GAD Time - Source (seconds)
3 (1 server, 1 worker)	6.20	1.97	1.309
4 (1 server, 2 workers)	7.58	4.19	2.039
5 (2 servers, 2 workers)	8.88	4.54	3.439
6 (2 servers, 3 workers)	9.91	5.11	4.155

Average time for call to UpdateSegmentList on first strip descriptor = 0.0000078 seconds

Machine: Lomond front end

Number of Processes	Sort Time (seconds)	Join time (seconds)	GAD Time - Source (seconds)
3 (1 server, 1 worker)	1.14	0.50	0.016
4 (1 server, 2 workers)	0.95	0.37	0.080
5 (2 servers, 2 workers)	1.55	1.47	0.400
6 (2 servers, 3 workers)	2.85	1.94	0.959

Average time for call to UpdateSegmentList on first strip descriptor = 0.0000083

Machine: Lomond back end

Number of Processes	Sort Time (seconds)	Join time (seconds)	GAD Time - Source (seconds)
3 (1 server, 1 worker)	0.66	0.33	0.032
4 (1 server, 2 workers)	0.60	0.16	0.072
5 (2 servers, 2 workers)	0.62	0.19	0.038
6 (2 servers, 3 workers)	0.55	0.17	0.037

Average time for call to UpdateSegmentList on first strip descriptor = 0.0000062

Machine: Work Station cluster

Number of Processes	Sort Time (seconds)	Join time (seconds)	GAD Time - Source (seconds)
4 (1 server, 2 workers)	0.82	0.47	0.691
5 (2 servers, 2 workers)	1.60	0.88	0.914
6 (2 servers, 3 workers)	1.22	0.63	0.896

Unfortunately no result could be obtained for 3 processors or the times to call UpdateSegmentList due to a software error.

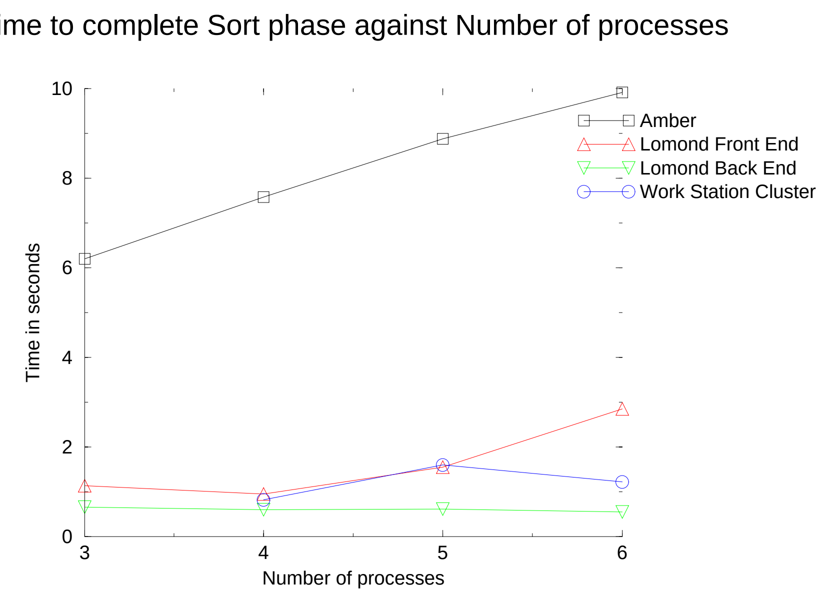

Figure 4: Results for Sort

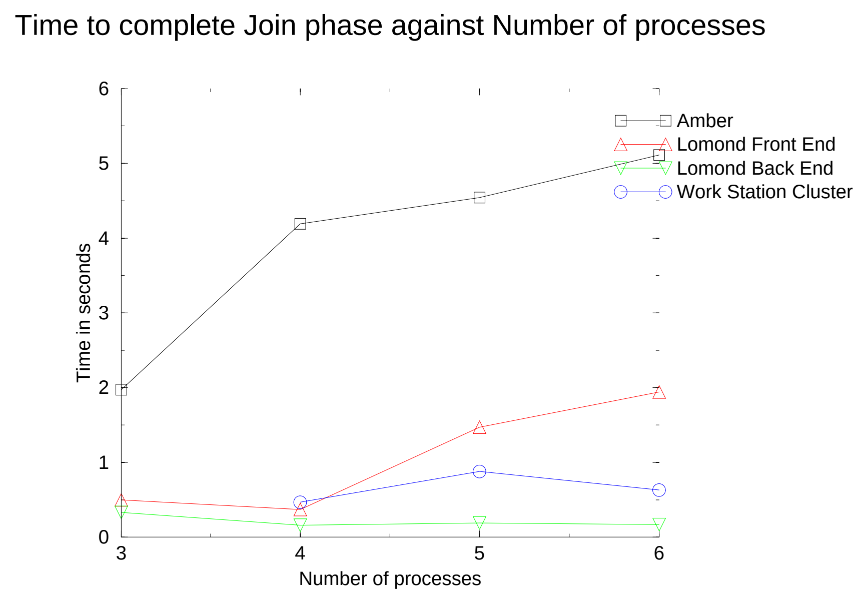

Figure 5: Results for Join

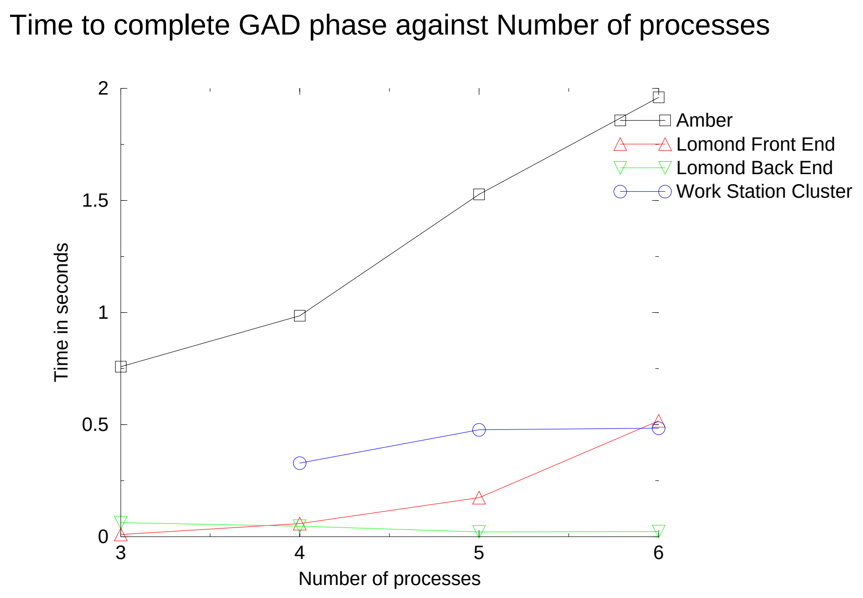

Figure 6: Results for GAD

5 Conclusion

As can be seen from the tables, the timing results for amber and the front end of lomond are fairly poor. This can be expected from amber, since it is a single processor machine. Work from previous SSP projects has also shown that the large amount of small messages that get passed in the Sort phase contributes to the poor scaling reflected in all the timings.

The back end of lomond gives some interesting results. The Sort and Join times do not improve in a monotonic manner, and the GAD phase gets faster after four processes. The results for Sort and Join are probably due to the balancing between increased communication and division of the tasks between processes. This may be attributable to the small dataset used, and could well disappear when division of the task between more processes yields more noticeable results on a larger set. The decrease in GAD source time is somewhat strange however, and would bear further investigation.

The cost of communication is exacerbated when working with the work station cluster, as they communicate across an ordinary network. This can be seen with the increase in time for Sort. The Join stage has a similar variation in time as that executed on the lomond back end, again probably due to the balancing of communication and workload, and the effects of using a small dataset.

The scaling of the program when a timing improvement occurs (such as on the E3500) is pretty poor, but the reasons for this have been given above. Hopefully larger datasets and the use of parallel I/O will improve the eventual parallel efficiency of the code, in conjunction with tests for pool splitting when the final implementation of rasterisation is completed.

5.1 Future Work

Unfortunately, due to timing constraints not as much of the functionality was implemented as I'd hoped, but hopefully the communication framework is now stable enough. Further implementation (completing RasteriseLine and its utility functions) only involves I/O and a small amount of GAD worker functionality, and so should be relatively straightforward to write. Testing on larger datasets would also be useful to investigate scaling and the heuristic for splitting the pool group on realistic rasterisation tasks.

6 The Summer Scholarship Programme

I applied to be a part of the EPCC SSP programme to get an insight into how research departments operate and specifically to gain experience of the state of the art in parallel computing. As part of my course next year I will undertake a large project, which involves implementing a library of parallel numerical algorithms on a simulated PRAM. This course has been a useful precursor to that, and in addition I have learnt a lot about high performance computing in general.

I would like to say thank you to Connor for supervising me, and to the EPCC for hosting the programme. I would like to thank the staff of EPCC and the other SSP students for making this an enjoyable ten weeks.

7 Appendix - Code

```
/* ****************************************************************** *
 *
 *
 *       Filename:      rasterise.c
 *
 *       Authors:       Connor Mulholland EPCC, Thomas Ashby SSP, Terry Sloan EPCC
 *
 *       Purpose:       Rasterisation
 *
 *       Used in:       V2R
 *                      EPSRC Parallel Geographical
 *                      Information Systems Project
 *                      Edinburgh Parallel Computing Centre/
 *                      University of Edinburgh Department of Geography
 *
 *       RCS: $Id: rasterise.c,v 1.12 1999/09/13 15:29:48 ashby Exp ashby $
 *
 * ****************************************************************** */

static char rcsid[]=
  "$Id: rasterise.c,v 1.12 1999/09/13 15:29:48 ashby Exp ashby $";

/* ****************************************************************** *
 *
 * $Log: rasterise.c,v $
 * Revision 1.12  1999/09/13 15:29:48  ashby
 * About to alter timing setup.
 *
 * Revision 1.11  1999/09/08 16:26:51  ashby
 * Update segment list should be working
 *
 * Revision 1.10  1999/08/27 08:57:38  ashby
 * Version that compiles, links, and
 * runs (after a fashion).
 *
 * Revision 1.9  1999/08/25 10:21:17  ashby
 * Hopefully done up to Update Segment List.
 * Will now try compiling.
 *
 * Revision 1.8  1999/08/16 12:35:35  ashby
 * Compiling to run totalview.
 *
 * Revision 1.7  1999/08/13 11:05:31  ashby
 * Hopefully done up to rasterise strip. Will now try compiling.
 *
 * Revision 1.6  1999/08/12 16:10:09  ashby
 * About to do rasterise function?
 *
 * Revision 1.5  1999/08/11 15:34:33  ashby
 * About to hack hoooj chunks out of rasterise/worker,
 * saving now to avoid catasrophy should failure occur.
 *
 * Revision 1.4  1999/08/10 11:49:47  ashby
 * This is the more complete shell. Am now working to get as much functioning as possible.
 *
 * Revision 1.2  1999/08/05 16:01:25  ashby
 * (Hopefully) done up to RasteriseWorker(); .
 *
 * Revision 1.1  1999/08/05 10:10:39  ashby
 * Initial revision
 *
 * Revision 1.1  1999/06/18 09:34:32  connor
 * Source code used in V2R phase.
 *
 *
 * ****************************************************************** */

/* ****************************************************************** *
 *
 * Included files
 *
 * ****************************************************************** */

/* #include <mpi.h> */

/* WARNING - This header was generated by Tom. */
#include "rasterise.h"

/* WARNING - not sure if this is the ultimate path to this header file, as
   the overlay internal header is stored in the po directory itself */
#include "../V2R/v2r-internal.h"

#include "../misc/debug.h"
/* #include "../misc/gis-internal.h" */
#include "../misc/partition.h"
/* #include "../include/gis.h" */
#include "../ntf/ntf.h"
#include "../vect_in/VectorInput.h"
/* #include "../gad/gad.h" */
#include "../gad/gad.mpi.h"
#include "../gad/gad-internal.h"
/* #include "../gad/gad-worker.h" */
#include "../gad/gad-worker-internal.h"
#include "../join/join.h"


/* ****************************************************************** *
 *
 * Global Variables
 *
 * ****************************************************************** */

static double overallTimeG;
static int thisRasteriseCommRankG;

/* ****************************************************************** *
 *
 * Initialise debugging stack
 *
 * ****************************************************************** */

DEBUG_DECL;


/* ****************************************************************** *
 *
 *      ~RasteriseSource()
 *
 * Effect : Rasterisation on Source process
 *
 * ****************************************************************** */
```

```
void RasteriseSource(char             *infile1,      /* 1st input file        */
                     char             *infile2,      /* 2nd input file        */
                     char             *outfile,      /* output file           */
                     RasteriseMethodT rType,         /* rasterisation  option *
                                                      * eg. Attribute area,   *
                                                      * weighted attribute    *
                                                      * area etc.             */
                     MPI_Comm         rasteriseComm, /* Communicator for      *
                                                      * all overlay           *
                                                      * processes             */
                     int              sourceRank,    /* source rank in        *
                                                      * rasteriseComm         */
                     int              nDatasets
                     )
{
  int                  mpiErr;                  /* MPI return value.     */
  int                  size;                    /* No of MPI processes   */
  VectorInputHeaderT   **infileHeader;          /* Input file headers    */
  ApplicArgsT          applicArgs;              /* Application arguments */
  int                  i;
  int                  xMin;
  int                  xMax;
  int                  yMin;
  int                  yMax;
  RealBoundingBoxT     boundingBox;
  int                  nMiniPartitions;
  PartitionSizeT       miniPartitionSize;

/* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< > >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */

  ENTRY("RasteriseSource");

  TRACE(DEBUG_ALL,
        printf("Process %d has entered RasteriseSource\n",
               sourceRank););


  /* ************************************************************** *
   * Check to make sure that there is only one dataset to avoid
   * vommiting later on.
   * ************************************************************** */

  if (nDatasets != 1){
     ERR_FATAL(GIS_INTERNAL_ERROR, "Number of datasets not equal to one.\n");
  }


  /* ************************************************************** *
   * Send rank of Source to all other processes in rasteriseComm MPI
   * communicator.
   * ************************************************************** */
  mpiErr = MPI_Comm_size(rasteriseComm, &size);
  if (mpiErr != MPI_SUCCESS)
      ERR_FATAL(mpiErr, "Error in MPI_Comm_size. \n");

  for (i = 0; i < size; i++)
  {
    if (i != sourceRank)
    {
      mpiErr = MPI_Send(&sourceRank,1,MPI_INT,i,0,rasteriseComm);

      if (mpiErr != MPI_SUCCESS)
        ERR_FATAL(mpiErr, "Error in MPI_Send. \n");
    }
  }

  /* ************************************************************** *
   * Get header for each input.
   * ************************************************************** */

  TRACE(DEBUG_ALL,
        printf("RasteriseSource: Getting input file header.\n"););

  infileHeader = (VectorInputHeaderT **)
                 gis_malloc(nDatasets *
                 sizeof(VectorInputHeaderT *));

  infileHeader[0] = VectorInputGetHeader(infile1);

  TRACE(DEBUG_ALL, printf("infileHeader[0]->nAttributes = %d.\n",
                          infileHeader[0]->nAttributes););

  /* ************************************************************** *
   * Check that the projections are compatible.
   *
   * It is not clear how this is going to be done since the VectorInput-
   * Header files do not include the neccessary field to test this.
   * ************************************************************** */

  /* ************************************************************** *
   * compute the bounding box from the input bounding box in the file
   * header.
   * ************************************************************** */
  /*
  boundingBox.xMin = infileHeader[0]->xMin;
  boundingBox.yMin = infileHeader[0]->yMin;
  boundingBox.xMax = infileHeader[0]->xMax;
  boundingBox.yMax = infileHeader[0]->yMax;
  */

  /* WARNING - We’re actually hard coding the bounding box size for the
               moment, and the cell sizes? - Tom.
  */
  boundingBox.xMin = (RealCoordT) -9999;
  boundingBox.yMin = (RealCoordT) -9999;
  boundingBox.xMax = (RealCoordT) 9999;
  boundingBox.yMax = (RealCoordT) 9999;

  /* ************************************************************** *
   * Construct the rasterisation arguments and drop the rasterisation
   * method into the wrapper function to be picked up by RasteriseStrip
   * later on.
   * ************************************************************** */

  /* WARNING - Randomly generated Cell size! */
  ConstructRasterisationArgs(&applicArgs,
                             infile1,
                             outfile,
                             boundingBox.xMin,
                       boundingBox.yMin,
                       (RealCoordT)0.1,
                       (RealCoordT)0.1,
                       60,
                       60);

  RasteriseMethodWrapper(TRUE, &rType);

  /* ******************************************************************** *
   * Write output file header - not too sure whether this is relevant
   * at this stage.
   * ******************************************************************** */

  /* ******************************************************************** *
   * Determine the mini partition size.  The partition size is
   * dependent upon this.
   * ******************************************************************** */

  nMiniPartitions = RASTERISE_GAD_N_MINI_PARTITIONS;

  /* DEBUG */
  printf("Number of mini partitions = %d\n", RASTERISE_GAD_N_MINI_PARTITIONS);

  miniPartitionSize = (boundingBox.yMax - boundingBox.yMin)/nMiniPartitions;

  /* ******************************************************************** *
   * Call VectorInputSource
   * ******************************************************************** */

    VectorInputSource(infile1,                  /* 1st input file        */
                      infile2,                  /* 2nd input file        */
                      outfile,                  /* output file           */
                      "GIS Package ID (GPID) ",  /* 1st attribute         */
                      NULL,                     /* 2nd attribute         */
                      &boundingBox,             /* operatioal area       */
                      &applicArgs,              /* Application arguments */
                      miniPartitionSize,        /* size of smallest work *
                                                 *  strip in y           */
                      nMiniPartitions,          /* Number of smallest    *
                                                 * in bounding box       */
                      &GADWorkContent,          /* function for work     *
                                                 * content in a strip    */
                      rasteriseComm,            /* Communicator for all  *
                                                 * overlay processes     */
                      sourceRank                /* source rank in        *
                                                 * overlayComm           */
                  );

  /* ******************************************************************** *
   * Return
   * ******************************************************************** */

  return;

/* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< End >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */

}

/* ********************************************************************** *
 *
 *      ~ConstructRasterisationArgs()
 *
 *
 * Effect :
 *
 * ********************************************************************** */

void ConstructRasterisationArgs(ApplicArgsT *applicArgs,
                                char        *infile1,
                                char        *outfile,
                                RealCoordT  x0,
                                RealCoordT  y0,
                                RealCoordT  xCellSize,
                                RealCoordT  yCellSize,
                                int         xCells,
                                int         yCells
                                )
{

  applicArgs->V2RArgs.infile = malloc(sizeof(char)*MAX_FILENAME_LENGTH);
  applicArgs->V2RArgs.outfile = malloc(sizeof(char)*MAX_FILENAME_LENGTH);

  strncpy(applicArgs->V2RArgs.infile, infile1, MAX_FILENAME_LENGTH);
  strncpy(applicArgs->V2RArgs.outfile, outfile, MAX_FILENAME_LENGTH);
  applicArgs->V2RArgs.areaRecsReq = TRUE;
  applicArgs->V2RArgs.x0 = x0;
  applicArgs->V2RArgs.y0 = y0;
  applicArgs->V2RArgs.xCellSize = xCellSize;
  applicArgs->V2RArgs.yCellSize = yCellSize;
  applicArgs->V2RArgs.nxCells = xCells;
  applicArgs->V2RArgs.nyCells = yCells;

}

/* ********************************************************************** *
 *
 *      ~RasterisePool()
 *
 *
 * Effect : Rasterisation on Pool process
 *
 * ********************************************************************** */

void RasterisePool(MPI_Comm   rasteriseComm,   /* Communicator for all    *
                                                * rasterisation processes */
                   int        nDatasets
                       )
{

  int                     sourceRank;      /* Rank of source process    */
  VectorInputArgsT        vectorArgs;      /* VectorInput arguments     */
  ApplicArgsT             applicArgs;      /* Application arguments     */
  int                     mpiErr;
  MPI_Status              status;
  VectorInputPoolReturnT poolGroupType;
  MPI_Comm                rasteriseWorkerComm;

  double                  rasterisePoolTime;
  int                     rasteriseCommRank;

/* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< > >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */

  ENTRY("RasterisePool");

  /* ******************************************************************** *
   *  Set the local clock ticking...
   * ******************************************************************** */

  rasterisePoolTime = MPI_Wtime();
```

```
  /* ******************************************************************** *
   *  Receive the rank of the Source process in the rasteriseComm MPI
   *  communicator.
   * ******************************************************************** */

  mpiErr = MPI_Recv(&sourceRank,
                    1,
                    MPI_INT,
                    MPI_ANY_SOURCE,
                    0,
                    rasteriseComm,
                    &status);

  if (mpiErr != MPI_SUCCESS)
    ERR_FATAL(mpiErr,"Error in MPI_Recv. \n");

  /* ******************************************************************** *
   *  Call VectorInputPool
   * ******************************************************************** */

  poolGroupType = VectorInputPool(rasteriseComm, sourceRank);

  /* ******************************************************************** *
   *  Depending on return value call the rasterise worker
   * ******************************************************************** */

  switch(poolGroupType.poolGroup)
  {

    case GEOM_ATTRIBUTE_SERVER_GROUP :
       break;

    case WORKER_GROUP :

       rasteriseWorkerComm = poolGroupType.comm;

       MPI_Comm_rank(rasteriseWorkerComm, &thisRasteriseCommRankG);

       /* ************************************************************** *
        * Extract ApplicArgs from VectorInputArgs (ie. poolGroupType.args).
        * ************************************************************** */

       vectorArgs = poolGroupType.args;
       applicArgs = vectorArgs.applicArgs;

       /* ************************************************************** *
        * Set global clock ticking
        * ************************************************************** */

       overallTimeG = MPI_Wtime();

       RasteriseWorker(rasteriseComm,
                       sourceRank,
                       rasteriseWorkerComm,
                       applicArgs,
                       nDatasets);
       break;

  }

  /* ******************************************************************** *
   * Output the local wall clock time.
   * ******************************************************************** */

  printf("Timing: Pool(%d): Function complete: Time to do rasterise pool = %f\n",
         thisRasteriseCommRankG,
         MPI_Wtime() - rasterisePoolTime);
  fflush(NULL);

  /* ******************************************************************** *
   * Return
   * ******************************************************************** */

  return;

/* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< End >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */

}

 /********************************************************************** *
 *
 *      ~RasteriseWorker()
 *
 *
 * Effect : Start up code for the Worker process in Rasterisation phase.
 *
 *
 * Arguments :
 *
 *            rasteriseComm    MPI communicator for all rasterisation
 *                             processes.
 *
 *            sourceRank       Rank of the source process in rasteriseComm.
 *
 *            workerCom        MPI Communicator for only worker processes.
 *
 *            ApplicArgs       Application arguments.
 *
 *
 * Note :  rasteriseComm comprises the Source and GeomAttributeServer
 *         processes, as well as all the workers.
 *
 *         workerComm excludes the data Source and GeomAttributeServer
 *         processes.
 *
 * ********************************************************************** */

void RasteriseWorker(MPI_Comm rasteriseComm,   /* Communicator for all     *
                                               * rasterisation processes  */
                     int      sourceRank,      /* Source Process Rank      */
                     MPI_Comm workerComm,      /* Communicator for only    *
                                               * worker processes         */
                     ApplicArgsT  applicArgs,  /* Applic arguments         */
                     int      nDatasets        /* No of datasets           */
                     ){

  GADWorkerArgsT         gadWorkerArgs;      /* GAD worker arguments       */
  void                   *strip;             /* include so hacked resource *
                                              * test can be accomplished   */

  GADWorkerAvailableT    workerAvailable;    /* structure containing       *
                                                worker resource value      */
  int                    mpiErr;
  GADStripDescMessageT   stripDesc,
                         *stripDescPtr;          /* GAD strip descriptor       */
  MPI_Status             status;
  int                    msg_src;            /* message source, gleaned    *
```

```
                                             * from status                 */
int                    i,j,k;               /* loop counters               */

MPI_Datatype           MPI_GADStripDescMessageT;
MPI_Datatype           MPI_GADStripSizeMsgT; /* MPI derived data type      */
MPI_Datatype           MPI_GADGeomTransferT;
MPI_Datatype           MPI_GeomValueYMaxRecT;
MPI_Datatype           MPI_Geom2ValuesYMaxRecT;

GADStripSizeMsgT       sizeMsg;             /* GAD size message            */
GADStripSizeMsgT       stripSizeMsg;        /* GAD size message            */
int                    dataMsgSize;         /* size of actual message,     *
                                             * gleaned from size message   */

GADGeomTransferT       dataMsg;             /* geom data mesage            */
GeomValueYMaxRecT      dataMsgGeomValue;    /* geom value Y max data       *
                                             * message                     */
Geom2ValuesYMaxRecT    dataMsgGeom2Values; /* geom two values Y max data   *
                                             * message                     */

GADGeomTransferT       *geomMessageIndex;   /* base of array for geom      *
                                             * messages                    */
GeomValueYMaxRecT      *geomValueMessageIndex;/* base of array for geom    *
                                               * value messages            */
Geom2ValuesYMaxRecT    *geom2ValuesMessageIndex;/* base of array for geom  *
                                                 * 2 values messages        */
int                    geomCount=0;         /* counter for geom messages   */
int                    geomValueCount=0;    /* counter for geom value      *
                                             * messages                    */
int                    geom2ValuesCount=0; /* counter for geom 2 values    *
                                             * messages                    */


double                 rasteriseWorkerTime;
int                    myRank;

/* added for debug 31/8/99 */
int intbuff;

/* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */

ENTRY("RasteriseWorker");

/* ********************************************************************* *
 * Start the local clock ticking...
 * ********************************************************************* */

rasteriseWorkerTime = MPI_Wtime();


/* ********************************************************************* *
 * Construct GADWorkerArgs.  This contains the arguments
 * required by GAD Workers.
 * ********************************************************************* */

GADWorkerConstructArgs(sourceRank, applicArgs, nDatasets);

/* ********************************************************************* *
 * Call GADWorkerInit
 *
 * This function receives a message containing the number of
 * GeomAttributeServes in the GAD phase.  It includes this number in
 * the GADWorkerArgsT structure.
 *
 * ********************************************************************* */

 GADWorkerInit();

 /* how the output file header gets written is to be determined yet.
  * Almost certainly from a "collator" process from in the workerComm
  * group. This is TSO's problem. The data for that header are to be
  * known to all worker processes */

 /* on exiting from this, nothing happens in operation-specific code
  * until applic_request_data is called */

/* ********************************************************************* *
 * Get the GADWorkerArgs.  This contains the arguments the
 * GAD workers use.
 * ********************************************************************* */

gadWorkerArgs = GADWorkerGetArgs();


/* ********************************************************************* *
 * Create the MPI type to send the worker available message.
 * ********************************************************************* */

mpiErr = MPI_Type_contiguous(1, MPI_INT, &MPI_GADWorkerAvailableT);
if (mpiErr != MPI_SUCCESS)
    ERR_FATAL(mpiErr, "Error in MPI_Type_contiguous.\n");

mpiErr = MPI_Type_commit(&MPI_GADWorkerAvailableT);
if (mpiErr != MPI_SUCCESS)
    ERR_FATAL(mpiErr, "Error in MPI_Type_commit.\n");


/* ******************************************************************* *
 * Initialise GAD MPI datatype used to receive the strip size        *
 * message, geom transfer message, geom value Y max rec T, and the
 * geom 2 values Y max rec T
 * ******************************************************************* */

GADInitMPITypesForStripDesc(&MPI_GADStripDescMessageT);
GADInitMPITypeForStripSizeMsg(&MPI_GADStripSizeMsgT);
GADInitMPITypesForTransferBuffer(&MPI_GADGeomTransferT,
                                 &MPI_GeomValueYMaxRecT,
                                 &MPI_Geom2ValuesYMaxRecT);

/* ********************************************************************* *
 * The following code works for a single strip of data, however, we
 * must loop over until no more strips are left.
 * ********************************************************************* */

do{  /* while (stripDesc.messageType = GAD_YRANGE) */

  /* ********************************************************************* *
   * Determine the amount of space available for a strip of data, ie.
   * the strip size.  In the first instance this will be a fixed size.
   * Ideally we would like to find out in some way how much available
   * memory the process has left to deal with. At present, we are not
   * sure how to resolve this but we should return to this.
   *
   * Try and allocate memory to hold the strip and return if failed to
   * allocate enough memory
   *
   * ********************************************************************* */
```

```
strip = gis_malloc(750); /* 750 is hard coding for Worker resource     */

if (strip == NULL)
  ERR_FATAL(GIS_NO_MEMORY, "Error in malloc (strip-applic_request_data).\n");

gis_free(strip);

/* ******************************************************************** *
 * Construct a GADWorkerAvailableT Message containing the
 * worker resource. The worker resource is based on the strip size.
 * ******************************************************************** */

workerAvailable = GAD_WORKER_RESOURCE_VALUE;

/* ******************************************************************** *
 * Send the GADWorkerAvailable message to the Source.
 * ******************************************************************** */

mpiErr = MPI_Send(&workerAvailable,
                  1,
                  MPI_GADWorkerAvailableT,
                  gadWorkerArgs.sourceRank,
                  GAD_WORKER_AVAILABLE_TAG,
                  rasteriseComm);

if (mpiErr != MPI_SUCCESS)
    ERR_FATAL(mpiErr, "Error in MPI_Send. \n");

/* ******************************************************************** *
 * Issue the receive for a strip descriptor message
 * ******************************************************************** */

mpiErr = MPI_Recv(&stripDesc,
                  1,
                  MPI_GADStripDescMessageT,
                  gadWorkerArgs.sourceRank,
                  GADs_DESC,
                  rasteriseComm,
                  &status);

if (mpiErr != MPI_SUCCESS)
    ERR_FATAL(mpiErr, "Error in MPI_Recv. \n");

/* ******************************************************************** *
 * Check the message source.
 * ******************************************************************** */

msg_src = status.MPI_SOURCE;

if (msg_src !=  gadWorkerArgs.sourceRank)
  ERR_FATAL(mpiErr,
            "RasteriseWorker : strip received  from incorrect sourceRank");

/* ******************************************************************** *
 * Process the strip descriptor according to its message type
 * ******************************************************************** */

if (stripDesc.messageType == GAD_YRANGE){

  /* ****************************************************************** *
   * Register the strip descriptor with GAD
   * ****************************************************************** */

  GADWorkerRegisterStripDesc(&stripDesc);

  /* ****************************************************************** *
   * Set up recv's to get the data from the GAD
   * GeomAttributeServers (GAServers).
   *
   * For each dataset, each GAServer will send 4 messages.  These are
   * two for the Geom records and two for the GeomValue records.
   * Of these two messages the first message is of fixed size and
   * it contains information on the size of the second message.
   * the second message contains the records themselves.
   *
   * The sizes of the buffers to hold these second messages is
   * derived from the the strip descriptor message.  The
   * strip descriptor message contains the total number of
   * geoms and the total number of vertices in a strip.
   *
   * ****************************************************************** */

  /* ************************************************************** *
   * Malloc the space to hold the messages
   * ************************************************************** */

  geomMessageIndex = (GADGeomTransferT *)
    gis_malloc(stripDesc.totalGeoms * sizeof(GADGeomTransferT));

  geomValueMessageIndex = (GeomValueYMaxRecT *)
    gis_malloc(2 * stripDesc.totalGeoms * sizeof(GeomValueYMaxRecT));

  geom2ValuesMessageIndex = (Geom2ValuesYMaxRecT *)
    gis_malloc(2 * stripDesc.totalGeoms * sizeof(Geom2ValuesYMaxRecT));


  for (i = 0; i < gadWorkerArgs.nGAServers; i++){

    /* ************************************************************** *
     * Receive a geom and a geomvalue strip size message
     * ************************************************************** */

    for (j = 0; j < 2; j++){  /* geom and geomValue parcels */

      /* ********************************************************** *
       * Issue the recv for the strip size.
       * ********************************************************** */

      mpiErr = MPI_Recv(&sizeMsg,              /* buffer to receive message */
                        1,                     /* size of buffer      */
                        MPI_GADStripSizeMsgT,  /* datatype, can be msg specific */
                        MPI_ANY_SOURCE,        /* rank - could be MPI_ANY_SOURCE *
                        GADs_STRIP,            /* message tag. See bits_of_gad.h *
                        rasteriseComm,         /* communicator */
                        &status);

      if (mpiErr != MPI_SUCCESS)
          ERR_FATAL(mpiErr, "Error in MPI_Recv. \n");


      /* TOM 11/8/99 - really not sure whether the following line is
         necessary */

      /* ************************************************************** *
       * Unpack the message from the buffer and check its contents
       * This message needs to indicate which dataset and which type of data
       * it refers to.  This is so that the message to receive the actual
       * data is set up correctly.
     * ******************************************************************* */

    stripSizeMsg = sizeMsg;

    /* ******************************************************************* *
     * Allocate the buffer to receive the strip data from this GAServer.
     * ******************************************************************* */

    dataMsgSize = stripSizeMsg.nRecs;

    /* ******************************************************************* *
     * Issue the recv for data msg.
     * ******************************************************************* */

    switch(stripSizeMsg.rectype){
      case REC_GEOM:

        mpiErr = MPI_Recv((geomMessageIndex + geomCount),
                          dataMsgSize,
                          MPI_GADGeomTransferT,
                          MPI_ANY_SOURCE,
                          REC_GEOM,
                          rasteriseComm,
                          &status);

        if (mpiErr != MPI_SUCCESS)
            ERR_FATAL(mpiErr, "Error in MPI_Recv. \n");

        geomCount += dataMsgSize;
        break;

      case REC_GEOMVALUEYMAX:

        mpiErr = MPI_Recv((geomValueMessageIndex + geomValueCount),
                          dataMsgSize,
                          MPI_GeomValueYMaxRecT,
                          MPI_ANY_SOURCE,
                          REC_GEOMVALUEYMAX,
                          rasteriseComm,
                          &status);

        if (mpiErr != MPI_SUCCESS)
            ERR_FATAL(mpiErr, "Error in MPI_Recv. \n");

        geomValueCount += dataMsgSize;

        break;

      case REC_GEOM2VALUESYMAX:

        mpiErr = MPI_Recv((geom2ValuesMessageIndex + geom2ValuesCount),
                          dataMsgSize,
                          MPI_Geom2ValuesYMaxRecT,
                          MPI_ANY_SOURCE,
                          REC_GEOM2VALUESYMAX,
                          rasteriseComm,
                          &status);

        if (mpiErr != MPI_SUCCESS)
            ERR_FATAL(mpiErr, "Error in MPI_Recv. \n");

        geom2ValuesCount += dataMsgSize;

        break;

      default:
        /* *************************************************************** *
         * Yack like a good'un
         * *************************************************************** */

        ERR_FATAL(GIS_INTERNAL_ERROR, "Unrecognised message type\n");

    } /* switch(stripSizeMsg.recType) */

  } /* for loop for geom and geomValue parcels */

} /* for loop for GAServers */

/* ******************************************************************* *
 * Check to make sure we got all the messages we thought we would,
 * and barf if we didn't.
 * ******************************************************************* */

/* WARNING - added for DEBUG ONLY!! */
stripDesc.totalGeoms = geomCount;

if(geomCount == stripDesc.totalGeoms){
  if(geomValueCount == stripDesc.totalGeoms * 2
     && geom2ValuesCount == 0){

    gis_free(geom2ValuesMessageIndex);

    /* ******************************************************************* *
     * Now rasterise what we've got. This function frees the space used
     * to hold the messages automatically once the parcels have been
     * filled.
     * ******************************************************************* */

    /*
    Rasterise(&stripDesc,
              &(applicArgs.V2RArgs),
              geomMessageIndex,
              geomValueMessageIndex,
              NULL);
    */

  }else if(geom2ValuesCount == stripDesc.totalGeoms * 2
           && geomValueCount == 0){

    /* ******************************************************************* *
     * Now rasterise what we've got. This function frees the space used
     * to hold the messages automatically once the parcels have been
     * filled.
     * ******************************************************************* */

    gis_free(geomValueMessageIndex);

    /*
    Rasterise(&stripDesc,
              &(applicArgs.V2RArgs),
              geomMessageIndex,
              NULL,
              geom2ValuesMessageIndex);
    */

  }else{

    /* ******************************************************************* *
       * Yack appropriately. "WRONG NUMBER OF GEOMVALUES REC'D"
       * ******************************************************************* */

      /*
      ERR_FATAL(GIS_INTERNAL_ERROR, "Wrong number of geomValues received. \n");
      */

      printf("Whoops, never mind.\n");

    }
  }else{
    /* ******************************************************************* *
     * Yack appropriately. "WRONG NUMBER OF GEOMS REC'D"
     * ******************************************************************* */

     ERR_FATAL(GIS_INTERNAL_ERROR, "Wrong number of geoms received. \n");

  }

  geomCount = 0;
  geomValueCount = 0;
  geom2ValuesCount = 0;

 } /* if (stripDesc.messageType == GAD_YRANGE) */

}while(stripDesc.messageType == GAD_YRANGE);

/* ******************************************************************* *
 * Output the local time.
 * ******************************************************************* */

/* Execute barrier first */
mpiErr = MPI_Barrier(MPI_COMM_WORLD);
if(mpiErr != MPI_SUCCESS)
  ERR_FATAL(mpiErr, "Error in MPI_Barrier\n");

printf("Timing: Worker(%d): Function complete: Time to do rasterise worker = %f\n",
       thisRasteriseCommRankG,
       MPI_Wtime() - rasteriseWorkerTime);
fflush(NULL);

}

/* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */


/* ********************************************************************* *
 *
 * Function: ~Rasterise
 *
 * Effect : Instigates the rasterisation of each strip.
 *
 * Called by : RasteriseWorker
 *
 * ********************************************************************* */

void Rasterise(GADStripDescMessageT  *stripDesc,
               V2RArgsT              *v2RArgs,
               GADGeomTransferT      *geomMessageIndex,
               GeomValueYMaxRecT     *geomValueMessageIndex,
               Geom2ValuesYMaxRecT   *geom2ValuesMessageIndex)

{
  RecordTypeT   geomValueType;
  ParcelT       *geomParcel;
  ParcelT       *geomValueParcel;

  double        rasteriseTime;

  /* ******************************************************************* *
   * Output global clock
   * ******************************************************************* */

  printf("Timing: Worker(%d): Time since worker began: Overall time to get to rasterise
         thisRasteriseCommRankG,
         MPI_Wtime() - overallTimeG);
  fflush(NULL);

  /* ******************************************************************* *
   * Start the local clock ticking...
   * ******************************************************************* */

  rasteriseTime = MPI_Wtime();

  /* ***************************************************************** *
   * Identify which type of geomValue the records are. If the
   * geom2ValuesMessageIndex is NULL pointer, then they must be
   * geomValueMessages.
   * ***************************************************************** */

  if (geom2ValuesMessageIndex == NULL)
    geomValueType = REC_GEOMVALUEYMAX;
  else
    geomValueType = REC_GEOM2VALUESYMAX;

  GADWorkerStripInit(&(geomParcel),
                     &(geomValueParcel),
                     stripDesc->totalGeoms,
                     stripDesc->totalGeoms *2,
                     geomValueType);

  /* ***************************************************************** *
   *  Fill the structure to contain the sorted geoms
   * ***************************************************************** */

  GADWorkerFillGeomParcel(stripDesc->totalGeoms,
                          geomMessageIndex,
                          geomParcel);

  /* ***************************************************************** *
   * Fill the structure to contain the sorted GeomValues
   * ***************************************************************** */

  GADWorkerFillValueParcel(stripDesc->totalGeoms*2,
                           geomValueMessageIndex,
                           geom2ValuesMessageIndex,
                           geomValueParcel,
                           geomValueType);

  RasteriseStrip(v2RArgs,
                 stripDesc,
                 geomParcel,
                 geomValueParcel,
                 geomValueType);
  /* ****************************************************************** *
   *  Free up the structures which have held the geoms and geomValues
   *  and clean out the strip descriptor.
   * ****************************************************************** */

  gis_free(geomMessageIndex);

  if (geomValueMessageIndex != NULL)
    gis_free(geomValueMessageIndex);


  if (geom2ValuesMessageIndex != NULL)
    gis_free(geom2ValuesMessageIndex);


  GADWorkerStripClose(geomParcel,
                      geomValueParcel);


 /* ******************************************************************* *
  * Output the local time.
  * ******************************************************************* */

  printf("Timing: Worker(%d): Function complete: Time to do rasterise = %f\n",
         thisRasteriseCommRankG,
         MPI_Wtime() - rasteriseTime);
  fflush(NULL);

  /* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */
}

/* ******************************************************************** *
 *
 * $Id: rasterise.c,v 1.12 1999/09/13 15:29:48 ashby Exp ashby $
 *
 *      ~RasteriseStrip()
 *
 * Effect:
 *
 * Called by:   Rasterise()
 *
 * Calls:       UpdateSegmentList(), RasteriseLine()
 *
 * Algorithm:   For  each  line of  the sub-raster  in turn,  this function
 *              updates   the segment list  and  then  fills  in the raster
 *              detail of  the line.
 *
 * I/O:         Writes an atom of raster data
 *
 * Side-effect: Memory dynamically allocated and freed.
 *
 * Returns:     None.
 *
 * ******************************************************************** */

void RasteriseStrip(V2RArgsT               *v2RArgs,
                    GADStripDescMessageT   *stripDesc,
                    ParcelT                *geomParcel,
                    ParcelT                *geomValueParcel,
                    RecordTypeT            geomValueType)
{
  SegmentListT       *asl;

  RealCoordT         yTop, yBottom, yCellSize;
  RealCoordT         yCells;
  RealCoordT         yUpper, yLower;

  int                line;

  RasteriseMethodT   method;
  GeomDescT          *firstGeomDesc;

  double             rasteriseStripTime;


  /* ******************************************************************* *
   * Output the global clock
   * ******************************************************************* */

  printf("Timing: Worker(%d): Time since worker began: Overall time to get to rasterise strip=%f\n",
         thisRasteriseCommRankG,
         MPI_Wtime() - overallTimeG);
  fflush(NULL);

  /* ******************************************************************* *
   * Start the local clock ticking...
   * ******************************************************************* */

  rasteriseStripTime = MPI_Wtime();


  /* ****************************************************************** *
   * Scan-fill each raster line
   * ****************************************************************** */

  asl = gis_malloc(sizeof(SegmentListT));

  /* ****************************************************************** *
   * Get strip y range, and the y cell size
   * ****************************************************************** */

  yTop = stripDesc->yRange.upper;
  yBottom = stripDesc->yRange.lower;
  yCellSize = v2RArgs->yCellSize;

  /* ****************************************************************** *
   * Calculate number of y cells in strip
   * ****************************************************************** */

  yCells = /* round */ ((yTop - yBottom) / yCellSize) ;

  /* ****************************************************************** *
   * Initialise variables to begin loop
   * ****************************************************************** */

  yUpper = yTop;
  asl->base = NULL;
  asl->end = NULL;

  firstGeomDesc = GADWorkerGetGeomDesc(geomParcel,
                                       geomValueParcel,
                                       geomValueType);

  /* ****************************************************************** *
   * Use the wrapper to get the method argument.
   * ****************************************************************** */
```

```
  method = RasteriseMethodWrapper(FALSE, NULL);

  for (line = 0; line < yCells; line ++)
    {
      /* ************************************************************** *
       * Delete old segments and add new ones to Active Segment
       * List (asl)
       * ************************************************************** */

      yLower = yUpper - yCellSize;

      asl = UpdateSegmentList(asl,
                              yUpper,
                              yLower,
                              geomParcel,
                              geomValueParcel,
                              geomValueType,
                              firstGeomDesc);


      /* ************************************************************** *
       * Rasterise scan-line y with arguments obtained from V2RArgs
       * and the method wrapper.
       * ************************************************************** */

      /*
        RasteriseLine(method,
        asl,
        v2RArgs->x0,
        v2RArgs->xCellSize,
        v2RArgs->nxCells,
        yUpper,
        yLower,
        firstGeomDesc);
      */

      yUpper = yLower;

    }

  gis_free(asl);

  /* flush and close output atom; */

  /* ****************************************************************** *
   * Output the local time
   * ****************************************************************** */

  printf("Timing: Worker(%d): Function complete: Time to do rasterise strip = %f\n",
         thisRasteriseCommRankG,
         MPI_Wtime() - rasteriseStripTime);
  fflush(NULL);
}

/* ******************************************************************** *
 *
 * $Id: UpdateSegmentList.pc,v 1.6 1994/03/25 12:32:41 tharding Exp tharding
 *
 *
 *      ~UpdateSegmentList()
 *
 * Effect:      Deletes segments above the current raster line and adds new
 *              ones that start on the current raster line.
 *
 * Called by:   RasteriseStrip()
 *
 * Calls:       GADWorkerGetGeomDesc(), CompareSegments() indirectly
 *              via qsort(),  GADWorkerFillGeomOutBuf
 *
 * Algorithm:   Geoms are extracted from the active buffer inBuf
 *              before deciding whether to add its first segment to
 *              the active segment list so there is usually an
 *              unprocessed pending geomDesc. This requires a geomDesc
 *              to be extracted when the function is first called for
 *              a given range.  Note that geomDesc will be set to NULL
 *              when the merge-tree output buffer is empty.
 *
 *              Each segment in asl must  have new x-extents calculated for
 *              it. This is trivial  with the signed xIncrement field. Then
 *              continuation  segments of segments in   asl that end on
 *              the current raster  line  are created  and inserted  in the
 *              correct position.  Then  each  segment in  the  new segment
 *              table  generated  by the  last   call of this  function  is
 *              checked to ensure that the segment’s position in the asl is
 *              still correct after the new x-extents have been calculated.
 *              Very rarely will segments  have to be  moved so instead  of
 *              using  qsort()  we do it   manually since already sorted is
 *              qsort’s worst case.
 *
 *              The old nst is then overwritten by new segments from inBuf,
 *              mallocing it  if it doesn’t   yet exist.  As  a segment  is
 *              formed out of a geom the correct geomValues are located via
 *              the hash-table, using the  flagged and unflagged GEOM_ID as
 *              the keys. The size of the table is dynamically increased by
 *              a certain  amount if necessary and  decreased  by a certain
 *              amount  if possible. The  tunable  amount prevents too much
 *              inefficient resizing. The nst  is  then sorted by  leftmost
 *              x-extent, using qsort(CompareSegments).
 *
 *              Then asl and nst  are merge-sorted: each  segment in asl is
 *              considered in turn, at  each point considering  whether the
 *              next  segment in nst  should be added  to asl at that point
 *              and  also whether to delete  a segment from  asl because it
 *              finishes above the current raster line.
 *
 * I/O:         None.
 *
 * Side-effect: Memory dynamically allocated and freed.
 *
 * Returns:     A pointer to the start of the ASL.
 *
 * ******************************************************************** */

SegmentListT *UpdateSegmentList(SegmentListT     *asl,
                                RealCoordT       yUpper,
                                RealCoordT       yLower,
                                ParcelT          *geomParcel,
                                ParcelT          *geomValueParcel,
                                RecordTypeT      geomValueType,
                                GeomDescT        *firstGeomDesc)
{

    static SegmentTableT  *nst = NULL;      /* New segment list */
    static SegmentTableT  *lastNST,
                          *tempNST;

    static GeomDescT      *geomDesc = NULL;
```

```
SegmentT            *newSegment,
                    *currentSegment,
                    *tempSegment,
                    *newSegments;

BoolT               notDead;
int                 i;
int                 hacktastic;

RealCoordT          xDiff;
RealCoordT          yDiff;
RealCoordT          yCellSize;

double              updateSegmentListTime;


/* ******************************************************************** *
 * Start the local clock ticking...
 * ******************************************************************** */

updateSegmentListTime = MPI_Wtime();

/* PREP */
/* ******************************************************************* *
 * If nst == NULL malloc the space for it and lastNST.
 * ******************************************************************* */

if(nst==NULL){

  /* ********************************************************************* *
   * Output global clock first time round
   * ********************************************************************* */

  printf("Timing: Worker(%d): Time since worker began: Overall time to get to updateSegmentList = %f\n",
         thisRasteriseCommRankG,
         MPI_Wtime() - overallTimeG);
  fflush(NULL);


  nst = gis_malloc(sizeof(SegmentTableT));
  nst->table = gis_malloc(NST_NUMSEG * sizeof(SegmentT));
  nst->size = 0;
  nst->sizeLimit = NST_NUMSEG;

  lastNST = gis_malloc(sizeof(SegmentTableT));
  lastNST-> table = NULL;
  lastNST-> size = 0;
}

if(geomDesc==NULL)
  geomDesc = firstGeomDesc;


/* MAIN */
/* ******************************************************************* *
 * Calculate x-extents for each segment in asl
 * ******************************************************************* */

currentSegment = asl->base;

while(currentSegment!=NULL){

  /* ************************************************************** *
   * If the segment ends above or on this rasterline, do the
   * nested test. Else calculate the new x-extents.
   * ************************************************************** */

  if(currentSegment->point2.y > yLower){

    /* ************************************************************** *
     * If the segment ends above the current raster line delete it,
     * else check for continuations and calculate x-extnets (special
     * case)
     * ************************************************************** */

    if(currentSegment->point2.y > yUpper){
      /* delete segment */

      /* remove from ASL */
      RemoveFromASL(currentSegment, asl);

      if(currentSegment->justAdded == TRUE){
        /* don't free mem. just yet, as other wise the lastNST
           checker will get crap instead of a segment */
        currentSegment->deleted = TRUE;
        currentSegment = currentSegment->aslNext;
      }else{
        tempSegment = currentSegment;
        currentSegment = currentSegment->aslNext;
        gis_free(tempSegment);
      }

    }else{
      /* check for continuations and calculate x-extents
         (special case - ends this line) */
      notDead = GatherContinuations(currentSegment,
                                    nst,
                                    yLower,
                                    yUpper - yLower);

      if(notDead == FALSE)
        GADWorkerDestroyGeomDesc(currentSegment->geomDesc);

      switch(SIGN(currentSegment->xIncrement)){
      case 1:
        currentSegment->xLeft = currentSegment->xRight;
        currentSegment->xRight = currentSegment->point2.x;
        break;
      case 0:
        /* in this case xLeft == xRight anyway */
        break;
      case -1:
        currentSegment->xRight = currentSegment->xLeft;
        currentSegment->xLeft = currentSegment->point2.x;
        break;
      default:
        ERR_FATAL(GIS_INTERNAL_ERROR, "Unexpected result from SIGN\n");
      }

      currentSegment = currentSegment->aslNext; /* get next in ASL */

    } /* if(currentSegment->point2.y > yUpper) */

  }else{

    /* calculate x-extents in normal way, provided the segment wasn't new last time */
    if(currentSegment->justAdded == FALSE){
      currentSegment->xRight += currentSegment->xIncrement;
      currentSegment->xLeft += currentSegment->xIncrement;
    }

    currentSegment = currentSegment->aslNext;

  } /* if(currentSegment->point2.y > yLower) */
} /* while(currentSegment!=NULL) */


/* ************************************************************ *
 * Now make sure new segments added last time are OK.
 * ************************************************************ */

currentSegment = lastNST->table;

/* ************************************************************ *
 * For each segment in lastNSL
 * ************************************************************ */

for(i=0; i<lastNST->size; ++i){
  if(currentSegment[i].deleted == FALSE){

    /* ************************************************************ *
     * Calculate the x-extents (special case for segments added
     * last time).
     * ************************************************************ */

    switch(SIGN(currentSegment[i].xIncrement)){
    case 1:
      currentSegment[i].xLeft = currentSegment[i].xRight;
      currentSegment[i].xRight += currentSegment[i].xIncrement;
      break;
    case 0:
      /* in this case xLeft == xRight anyway */
      break;
    case -1:
      currentSegment[i].xRight = currentSegment[i].xLeft;
      currentSegment[i].xLeft += currentSegment[i].xIncrement;
      break;
    default:
      ERR_FATAL(GIS_INTERNAL_ERROR, "Unexpected result from SIGN\n");
    }

    /* ************************************************************ *
     * Update the parent pointer so it points to something useful
     * rather than junk in the lastNST
     * ************************************************************ */

    if(asl->base != asl->end){

      /* If not head of asl... */
      if(currentSegment[i].aslPrev !=NULL)
        currentSegment[i].parent = currentSegment[i].aslPrev;
      /* ...or if not tail... */
      else if(currentSegment[i].aslNext != NULL)
        currentSegment[i].parent = currentSegment[i].aslNext;
      /* ...otherwise asl has only one element (drop through),
         or it is empty and something has gone
         badly wrong as the elements in the lastNST should
         have been inserted -> yack */
      else if(asl->base == NULL)
        ERR_FATAL(GIS_INTERNAL_ERROR, "Last NST elements not inserted correctly\n")

      /* ************************************************************ *
       * If it is no longer in the right place, remove it and
       * re-insert it.
       * ************************************************************ */

      if(currentSegment[i].aslPrev != NULL &&
         currentSegment[i].xLeft < currentSegment[i].aslPrev->xLeft){
        RemoveFromASL((currentSegment + i), asl);
        InsertInASL(asl, 1, (currentSegment + i));
      }
    }

    /* ************************************************************ *
     * Unmark 'justAdded' so segment is caught by general case
     * x-extents calculation next time.
     * ************************************************************ */

    currentSegment[i].justAdded = FALSE;

  }else{
    gis_free(currentSegment);
  }

} /* for(i=0; i<lastNST->size; ++i){ */


/* **************************************************************** *
 * Bit of a nast hack - rather than having 2 new segment tables,
 * I am going to remember how many got added above (ie. to the start
 * of the table) and I will IGNORE them when doing the quicksort for
 * segments added from new geoms and the following merge into the
 * active segment list.
 * **************************************************************** */

hacktastic = nst->size;

/* **************************************************************** *
 * Add the new segments from geoms starting on this line, and any
 * continuations to the NST.
 * **************************************************************** */

while((geomDesc != NULL &&
       geomDesc != NO_MORE_GEOMDESCS)
      &&
      (geomDesc->geom->points[0].y > yLower ||
       geomDesc->geom->points[geomDesc->geom->nPoints-1].y > yLower)){

  /* build first segment */
  newSegment = BuildSegment(geomDesc, 0, NULL, nst);

  /* check to see whether it stops on this line */
  if(newSegment->point2.y > yLower){

    /* calculate x-extents - special case (stumpy segment) */
    xDiff = newSegment->point1.x - newSegment->point2.x;

    switch(SIGN(xDiff)){
    case -1:
      newSegment->xLeft = newSegment->point1.x;
      newSegment->xRight = newSegment->point2.x;
      newSegment->xIncrement = xDiff; /* added for completeness - probably
                                         redundant */
      break;
```

```
        case 0:
          newSegment->xLeft = newSegment->point1.x;
          newSegment->xRight = newSegment->point1.x;
          newSegment->xIncrement = 0; /* added for completeness - probably
                                         redundant */
          break;
        case 1:
          newSegment->xLeft = newSegment->point2.x;
          newSegment->xRight = newSegment->point1.x;
          newSegment->xIncrement = xDiff; /* added for completeness - probably
                                             redundant */
          break;
        default:
          ERR_FATAL(GIS_INTERNAL_ERROR, "Unexpected return value from SIGN\n");
        }

        /* get continuations */
        notDead = GatherContinuations(newSegment,
                                      nst,
                                      yLower,
                                      yUpper - yLower);

        if(notDead==FALSE)
          GADWorkerDestroyGeomDesc(newSegment->geomDesc);


      }else{
        /* calculate x-extents for new segment (special case - start of segment) */
        xDiff = (newSegment->point1.x - newSegment->point2.x);
        yCellSize = yUpper - yLower;

      switch(SIGN(xDiff)){
      case -1:
        yDiff = (newSegment->point1.y - newSegment->point2.y);
        newSegment->xLeft = newSegment->point1.x;
        newSegment->xRight = newSegment->point1.x
                               - (xDiff / yDiff) * (newSegment->point1.y - yLower);
        newSegment->xIncrement = -(xDiff / yDiff) * yCellSize;
        break;
      case 0:
        newSegment->xLeft = newSegment->point1.x;
        newSegment->xRight = newSegment->point2.x;
        newSegment->xIncrement = 0;
        break;
      case 1:
        yDiff = (newSegment->point1.y - newSegment->point2.y);
        newSegment->xLeft = newSegment->point1.x
                              - (xDiff / yDiff) * (newSegment->point1.y - yLower);
        newSegment->xRight = newSegment->point1.x;
        newSegment->xIncrement = -(xDiff / yDiff) * yCellSize;
        break;
      default:
        ERR_FATAL(GIS_INTERNAL_ERROR, "Unexpected return value from SIGN\n");
      }

      /* get the next geom descriptor */
      geomDesc = GADWorkerGetGeomDesc(geomParcel,
                                      geomValueParcel,
                                      geomValueType);

      }

    } /* while(geomDesc != NULL &&
          geomDesc->geom->points[geomDesc->geom->nPoints -1].y > yLower) */

    /* ******************************************************************* *
     * Quick sort segments in the new segment table added from new
     * geom descriptors (see hack above).
     * ******************************************************************* */

    qsort((void *)(nst->table + hacktastic),
          (size_t)(nst->size - hacktastic),
          sizeof(SegmentT),
          &CompareSegments);

    /* ******************************************************************* *
     * Use InsertInASL to effectively merge-sort the NST with the ASL
     * ******************************************************************* */

    if((nst->size) != 0){

      /* copy the table */
      newSegments = gis_malloc(nst->size * sizeof(SegmentT));
      memcpy(newSegments, nst->table, (nst->size * sizeof(SegmentT)));

      /* use parent pointer to effect a merge sort of segments from new
         geom descriptors */
      if(nst->size - hacktastic != 0){ /* annoying test, but necessary to avoid
                                          segmentation faults */
        newSegments[hacktastic].parent = asl->end;
        for(i=hacktastic + 1; i<nst->size; ++i)
          newSegments[i].parent = (newSegments + i-1);
      }

      /* insert ALL of NST and update lastNST */
      lastNST->table = InsertInASL(asl,
                                   nst->size,
                                   newSegments);
      lastNST->size = nst->size;

      /* clear the NST */
      nst->size = 0;
    }else
      /* update lastNST */
      lastNST->size = 0;


    /* ******************************************************************** *
     * Output the local time.
     * ******************************************************************** */

    printf("Timing: Worker(%d): Function complete: Time to do update segment list = %f\n",
           thisRasteriseCommRankG,
           MPI_Wtime() - updateSegmentListTime);
    fflush(NULL);

    /* ******************************************************************* *
     * Return the pointer to the new ASL.
     * ******************************************************************* */

    return asl;
}


/* *********************************************************************** *
 *
 * End of UpdateSegmentList.pc
 *
```

```
 * ******************************************************************* */

/* ******************************************************************* *
 *
 * Utility recursive function which checks for continuations after a
 * segment and calls build if there are. If the new segment stops before
 * the bottom of the raster line, the function recurses.
 *
 * ******************************************************************* */

BoolT GatherContinuations(SegmentT        *currentSegment,
                          SegmentTableT   *nst,
                          RealCoordT      yLower,
                          RealCoordT      yCellSize)
{
  SegmentT   *newSegment;
  BoolT      notDead;
  RealCoordT xDiff;
  RealCoordT yDiff;

  /* Check for continuations */
  if(currentSegment->number < (currentSegment->geomDesc->geom->nPoints - 2)){

    /* build it */
    newSegment = BuildSegment(currentSegment->geomDesc,
                              currentSegment->number +1,
                              currentSegment,
                              nst);

    /* if it stops before the end of this raster line, recurse on it */
    if(newSegment->point2.y > yLower){

      /* Recursive Case */
      notDead = GatherContinuations(newSegment,
                                    nst,
                                    yLower,
                                    yCellSize);

      /* calculate x-extents for stumpy segment (special case) */
      xDiff = (newSegment->point1.x - newSegment->point2.x);
      switch(SIGN(xDiff)){
      case -1:
        newSegment->xLeft = newSegment->point1.x;
        newSegment->xRight = newSegment->point2.x;
        newSegment->xIncrement = 0;
        break;
      case 0:
        newSegment->xLeft = newSegment->point1.x;
        newSegment->xRight = newSegment->point2.x;
        newSegment->xIncrement = 0;
        break;
      case 1:
        newSegment->xLeft = newSegment->point2.x;
        newSegment->xRight = newSegment->point1.x;
        newSegment->xIncrement = 0;
        break;
      default:
        ERR_FATAL(GIS_INTERNAL_ERROR, "Unexpected return value from SIGN\n");
      }

      return notDead;

    }else{
      /* Base Case 1 */

      /* calculate x-extents for normal segment (special case - start of segemnt) */
      xDiff = (newSegment->point1.x - newSegment->point2.x);

      switch(SIGN(xDiff)){
      case -1:
        yDiff = (newSegment->point1.y - newSegment->point2.y);
        newSegment->xLeft = newSegment->point1.x;
        newSegment->xRight = newSegment->point1.x
                               - (xDiff / yDiff) * (newSegment->point1.y - yLower);
        newSegment->xIncrement = -(xDiff / yDiff) * yCellSize;
        break;
      case 0:
        newSegment->xLeft = newSegment->point1.x;
        newSegment->xRight = newSegment->point2.x;
        newSegment->xIncrement = 0;
        break;
      case 1:
        yDiff = (newSegment->point1.y - newSegment->point2.y);
        newSegment->xLeft = newSegment->point1.x
                              - ((xDiff / yDiff) * (newSegment->point1.y - yLower));
        newSegment->xRight = newSegment->point1.x;
        newSegment->xIncrement = -(xDiff / yDiff) * yCellSize;
        break;
      default:
        ERR_FATAL(GIS_INTERNAL_ERROR, "Unexpected return value from SIGN\n");
      }

      /* segment ends below this rasterline (ie. geom not dead yet) */
      return TRUE;
    }

  }else{
    /* Base Case 2 */

    /* all segements end on this line - kill geom */
    return FALSE;
  }
}



/* ****************************************************************** *
 *
 * Function: BuildSegment
 *
 * Gets the relevant segment (denoted by number) from the geom pointed
 * to by geomDesc, and makes it in the next available slot in the table
 * of the new segment table structure (nst). Reallocs the nst if there
 * are no slots left. Also reallocs after the function is complete
 * if possible, to save space.
 * ****************************************************************** */

SegmentT *BuildSegment(GeomDescT      *geomDesc,
                       int            number,
                       SegmentT       *parent,
                       SegmentTableT  *nst)
{
  int index;
  int index2;
  /* jiggery pokery to get round fact that points can be ordered
     either way in a geom */
  if(geomDesc->geom->points[0].y > geomDesc->geom->points[geomDesc->geom->nPoints-1].y){
    index = number;
    index2 = number + 1;
  }else{
    index = geomDesc->geom->nPoints-1 - number;
    index2 = geomDesc->geom->nPoints-2 - number;
  }


  /* realloc the table if it isn't big enough */
  if(nst->size == nst->sizeLimit){
    nst->table = realloc((void *)nst->table, (nst->sizeLimit + NST_INCSEG)*sizeof(SegmentT));
    nst->sizeLimit += NST_INCSEG;
  }

  /* initialise segment */
  nst->table[nst->size].geomDesc = geomDesc; /* pointer to parent geomDesc */
  nst->table[nst->size].point1 = geomDesc->geom->points[index];
  nst->table[nst->size].point2 = geomDesc->geom->points[index2];
  nst->table[nst->size].parent = parent;
  nst->table[nst->size].number = number;
  nst->table[nst->size].justAdded = TRUE;
  nst->table[nst->size].deleted = FALSE;
  nst->table[nst->size].aslNext = NULL;
  nst->table[nst->size].aslPrev = NULL;
  nst->table[nst->size].cslNext = NULL;

  nst->size++;

  /* realloc the table to save space if possible (as long as it isn't min size already) */
  if(nst->size < nst->sizeLimit - NST_INCSEG && nst->sizeLimit > NST_NUMSEG){
    nst->table = realloc((void *)nst->table, (nst->sizeLimit - NST_INCSEG)*sizeof(SegmentT));
    nst->sizeLimit -= NST_INCSEG;
  }

  /* return a pointer to the segement just initialised in nst */
  return (nst->table + nst->size-1);
}



/* ******************************************************************* *
 *
 * Function: InsertInASL
 *
 * Inserts 'numOfSegments' number of segments into the active segement
 * list pointed to by asl. The newSegments pointer is treated as the
 * base of a contiguous array if numOfSegments is > 1. The segments
 * are assumed to have a member 'parent' that points to a member in
 * the asl or is NULL if the asl is empty.
 * ******************************************************************** */


SegmentT *InsertInASL(SegmentListT  *asl,
                      int           numOfSegments,
                      SegmentT      *newSegments)
{
  int       i;
  SegmentT  *current;


  /* for each segment in table (usually nst table) */
  for(i=0; i<numOfSegments; ++i){

    /* get its parent */
    current = newSegments[i].parent;

    /* provided there is a parent (=== to saying the asl is NOT empty) */
    if(current != NULL){

      /* check to see which way we are scanning */
      if(newSegments[i].xLeft < current->xLeft){

        /* scan left (do{}while construct used to avoid redundant first test) */
        do current = current->aslPrev;
        while(current != NULL && newSegments[i].xLeft < current->xLeft);

      }else{

        /* scan right */
        while(current->aslNext != NULL && newSegments[i].xLeft >= current->aslNext->xLeft)
          current = current->aslNext;

      }

      /* check for 'head of list' condition */
      if(current==NULL){
        newSegments[i].aslNext = asl->base;
        newSegments[i].aslPrev = NULL;
        asl->base = (newSegments + i);
      }else{
        /* insert normally */
        newSegments[i].aslNext = current->aslNext;
        newSegments[i].aslPrev = current;
        current->aslNext = (newSegments + i);
      }

      /* if not 'end of list' complete join */
      if(newSegments[i].aslNext != NULL)
        newSegments[i].aslNext->aslPrev = (newSegments + i);
      else
        /* otherwise update the asl */
        asl->end = (newSegments + i);
    }else{
      /* asl previously empty, so insert first element */
      /* [NB Only ever occurs with first element of NST - after that ASL is
             not empty. Be careful with parent pointers!] */
      asl->base = newSegments;
      asl->end = newSegments;
    }

  }/* for(i=0; i<numOfSegments; ++i){ */

  return newSegments;
}

/*
  Describe RemoveFromASL
*/

void RemoveFromASL(SegmentT      *segment,
                   SegmentListT  *asl)
{

  /* normal case */
  if(segment->aslPrev != NULL
     && segment->aslNext != NULL){
    segment->aslNext->aslPrev = segment->aslPrev;
    segment->aslPrev->aslNext = segment->aslNext;

  /* head of list */
  }else if(segment->aslPrev == NULL){
    asl->base = segment->aslNext;
    segment->aslNext->aslPrev = NULL;

  /* end of list */
  }else if(segment->aslNext == NULL){
    asl->end = segment->aslPrev;
    segment->aslPrev->aslNext = NULL;

  /* single element */
  }else if(segment->aslPrev == NULL
     && segment->aslNext == NULL){
    asl->base = NULL;
    asl->end = NULL;
  }

}

/* ******************************************************************** *
 *
 * $Id: rasterise.c,v 1.12 1999/09/13 15:29:48 ashby Exp ashby $
 *
 *      ~CompareSegments()
 *
 * Effect:      Compares  the  minimum  x  extents  of  two  segments on  a
 *              scan-line.
 *
 * Called by:   qsort()
 *
 * Calls:
 *
 * Algorithm:   Uses the  difference between the  leftmost  x-values of the
 *              two segments on the current  raster line, to decide whether
 *              their order should be swapped.
 *
 * I/O:         None.
 *
 * Side-effect: None.
 *
 * Returns:     The sign of the difference  between the xLefts of the first
 *              and second segments.
 *
 * ********************************************************************* */

int CompareSegments(SegmentT *segment1, SegmentT *segment2)
{
    return SIGN(segment1->xLeft - segment2->xLeft);         /* Ascending */
}

/* ******************************************************************** *
 *
 * End of CompareSegments.pc
 *
 * ********************************************************************* */

/*
   Describe RasteriseMethodWrapper
*/


RasteriseMethodT static RasteriseMethodWrapper(BoolT             construct,
                                               RasteriseMethodT  *rMethod
                                               )
{
  static RasteriseMethodT  method;

  if(construct==TRUE && rMethod!=NULL){
    method = *rMethod;
    return GIS_V2R_ATTR_AREA;        /* This is a nonsense value which gets
                                        chucked away */
  }else if(construct==FALSE && rMethod==NULL){
    return method;
  }else{
    ERR_FATAL(GIS_INTERNAL_ERROR, "RasteriseMethodWrapper called with wrong arguments.
  }
}
```

References

[1] Tor Bernhardsen. *Geographic Information Systems*. Viak IT, Norway, 1992.

[2] Donna J. Peuquet, Duane F. Marble, editor. *Introductory readings in Geographic Information Systems*. Taylor and Francis, 1990.

[3] Richard Healey, Steve Dowers, Bruce Gittings and Mike Mineter, editor. *Parallel Processing Algorithms for GIS*. Taylor and Francis, 1998.

[4] Connor Mulholland. Rasterisation: Detailed design update. Technical report, EPCC, 1999.

Thomas Ashby
Final year joint honours degree, Artificial Intelligence and Computer Science
University of Edinburgh
e-mail: T.J.Ashby@sms.ed.ac.uk

Supervisor - Connor Mulholland