On the design and implementation of a parallel, object-oriented, image processing toolkit *

Chandrika Kamath, Chuck H. Baldwin, Imola K. Fodor, and Nu A. Tang

Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA 94551

ABSTRACT

Advances in technology have enabled us to collect data from observations, experiments, and simulations at an ever increasing pace. As these data sets approach the terabyte and petabyte range, scientists are increasingly using semi-automated techniques from data mining and pattern recognition to find useful information in the data. In order for data mining to be successful, the raw data must first be processed into a form suitable for the detection of patterns. When the data is in the form of images, this can involve a substantial amount of processing on very large data sets.

To help make this task more efficient, we are designing and implementing an object-oriented image processing toolkit that specifically targets massively-parallel, distributed-memory architectures. We first show that it is possible to use object-oriented technology to effectively address the diverse needs of image applications. Next, we describe how we abstract out the similarities in image processing algorithms to enable re-use in our software. We will also discuss the difficulties encountered in parallelizing image algorithms on massively parallel machines as well as the bottlenecks to high performance. We will demonstrate our work using images from an astronomical data set, and illustrate how techniques such as filters and denoising through the thresholding of wavelet coefficients can be applied when a large image is distributed across several processors.

Keywords: distributed algorithms, image processing, parallel performance, wavelets, object-oriented software

1. INTRODUCTION

Image processing plays an important role in the analysis of images using data mining and pattern recognition techniques. Image data is usually available in its raw form as pixel values. However, the object or pattern of interest is often at a higher level such as a galaxy, a road, or a face. As a result, higher level features that are representative of the pattern must first be extracted from the image, prior to pattern recognition. This task can be very compute intensive when the images are large, either in size, or in number, or both. In such cases, parallel processing can play an important role in reducing the turnaround time.

In this paper, we show how we can use object-oriented techniques, in conjunction with parallel processing, to design and implement an efficient toolkit for image processing tasks. In Section 2, we first describe the process of data mining for extracting useful information from data. We identify several image processing tasks that can help us in feature extraction. In Section 3, we illustrate how object-oriented techniques can be used to abstract out commonalities in different image processing tasks. In Section 4, we discuss the approach we have taken to implement some of these tasks in parallel. We describe our on-going efforts in the Sapphire project[1] to re-use our code, both in the computation and communication parts of the software. Section 5 briefly describes the task of denoising image data by thresholding wavelet coefficients. In Section 6, we provide some preliminary results of our work in denoising an astronomical data set. We conclude in Section 7 with a summary.

*Accepted for publication in Parallel and Distributed Methods for Image Processing IV, at The International Society for Optical Engineering (SPIE) Annual Meeting, July 2000

Further author information: (Send correspondence to I.K.F.)

C.K.: E-mail: kamath2@llnl.gov

C.H.B.: E-mail: baldwin5@llnl.gov

I.K.F.: fodor1@llnl.gov

N.A.T.: tang10@llnl.gov

2. ROLE OF IMAGE PROCESSING IN DATA MINING

Data mining is a process concerned with uncovering patterns, associations, anomalies, and statistically significant structures in data.[2–4] It is an iterative and interactive process involving data preparation, search for patterns, knowledge evaluation, and refinement of the process based on input from domain experts. In the data preparation stage, we extract features from the raw data that are representative of the data and relevant to the problem being solved. This step can consist of several stages such as the use of sampling or multi-resolution to reduce the size of the data, denoising of the data, feature extraction, and dimension reduction to reduce the number of features (or dimension) of the problem. This critical first step can frequently take up to 90% of the total time for data mining in moderate-sized data sets.

The application of data mining to image data typically involves the use of image processing techniques in the data preparation step. The compute intensive nature of these tasks, especially when the data set is very large, makes these tasks ideal for parallel processing. If the data set consists of a large number of small to moderate size images, an obvious use of parallel processors would be to assign one or more images to each processor. However, if each image is itself very large, we may want to use parallel processing within an image. To do this efficiently can be very challenging.

Image processing techniques that are commonly used in mining image data include image registration, enhancement, denoising, segmentation, edge detection, feature extraction and multi-resolution analysis.[5–7] In this paper, we will illustrate our approach to parallel implementation using wavelets and wavelet denoising. These are appropriate operations to consider as they are composed of other operations that occur in several image processing tasks. For example, we can consider wavelets and multi-resolution analysis in terms of filters banks composed of high- and low-pass filters. Efficient implementation of filters in the context of wavelets will therefore help several other operations such as edge detection and smoothing. In addition, certain ideas developed in the context of filters, can also be applied in other areas such as morphological image processing.

We next briefly describe how object-oriented techniques can help us to abstract out the commonalities in image processing operations. Through the use of object-oriented design and programming, we can support several different input data formats in a user-friendly interface. In addition, for the parallel implementation of our algorithms, we can identify and isolate the parallel processing tasks that are common across several operations. The resulting software re-use can enable us to easily enhance the functionality of our software.

3. BENEFITS OF OBJECT-ORIENTED DESIGN AND PROGRAMMING

In the last decade, there has been an increasing interest in using object-oriented paradigms for the development of software in scientific applications.[8] This approach is attractive as it supports well-defined mechanisms for a modular design, re-use of code, data abstractions, and the creation of flexible software that can easily be enhanced to support new applications as well as solution techniques. While object-oriented applications may initially be more abstract, difficult to understand and implement, and possibly slower, they do provide the means of addressing complex problems through step-wise software development. We briefly illustrate this through a simple example.

The data that is input to an image processing application can vary in the format used for storing the data. Often, several different formats are supported even within a single application domain such as astrophysics. Sometimes, even a single data format can support multiple options. For example, the FITS format used in astronomy[9] can store one-, two-, or three-dimensional, real or integer data. Given the numerous formats currently in use in the image processing community, it is impractical and cumbersome to provide support for all of them. A simple solution to this problem is to first convert the input format into a standard format and then operate only on the standard format.

Figure 1 describes the class hierarchy we use for implementing this solution. All the domain specific data formats, such as FITSData and HDFData are derived from a base class, DomainData. The base class provides the interface for common operations such as reading and writing an input data file. Each of the derived classes implements the operation based on the specifics of the data format. As a result, much of the detail of a data format is hidden from the user, providing a friendlier interface. The RegData class is our internal standard data format for one-, two-, and three-dimensional data. It is templated on the type of data — floats, doubles, or integers. The template mechanism allows us to easily support operations for the case where the identical code is executed on different data types. It also allows us to easily add data types as the need for them arises, without having to create a new version of the code for the new data type.

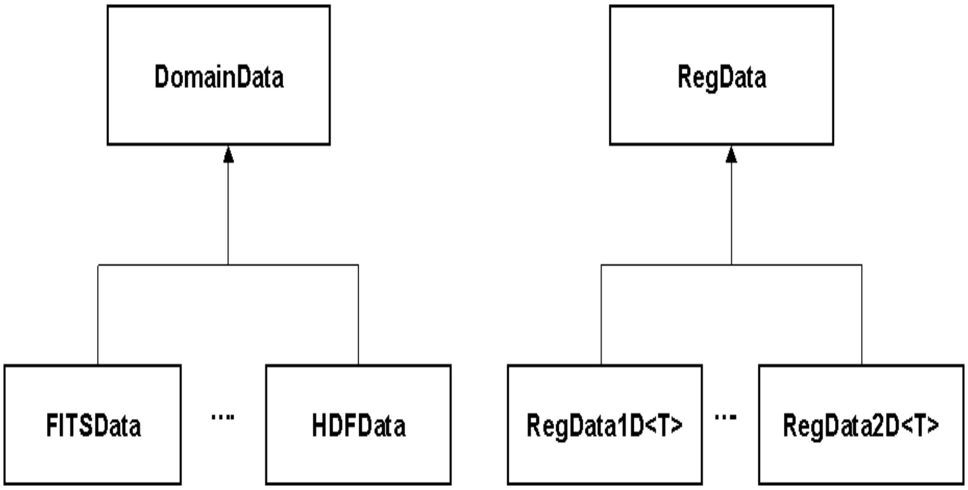

Figure 1. The class hierarchy that enables us to support different data formats

There are other areas where we can benefit by abstracting out the commonalities in image processing operations. For example, many techniques in image processing can be expressed as operations among local collections of pixels (or geometric regions). Linear filters[6] and discrete wavelet operators,[10] where a linear transformation is applied to neighboring pixels in the specified region, are examples of such techniques. Other examples where the transformation depends on the local data are non-linear filters and morphological operators. We distinguish between transformations that have associated values and those which do not. The former are defined by the *Stencil* class and the latter by the *Neighborhood* class. The geometric regions associated with both are formulated by the user as index offsets from an implied origin. An object in each class is configured with these offsets. In the case of a *Stencil* class, there are also associated values that are templated on the data type. The *Stencil* operator can be used in filtering, wavelet operators, or any other operation where the values can be a-priori associated with specific pixels in a region. The *Neighborhood* is used in non-linear filtering, morphological operators, and other contexts where the user is unable to associate values with specific pixels in a region – either because values do not exist or they vary in space or time, and therefore cannot be hard coded into the object. Figure 2 is an example of a simple 3 by 3 *Neighborhood*.

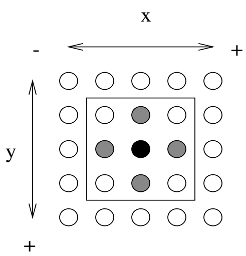

Figure 2. Example of a 3 by 3 *Neighborhood*

In this example, the implied origin is the black circle $\bullet$. It, along with the gray circles, represents the specific collection of pixels that is the geometric region of the *Neighborhood*. Note that the origin is part of the *Neighborhood*. The outline (or the *bounding box*) represents the maximum extents of the collection of pixels that form the *Neighborhood*. The region is "encoded" in the example *Neighborhood* object by the following collection of (x, y) offsets:

$$\{(0,-1),(-1,0),(0,0),(1,0),(0,1)\}.$$

Our object-oriented software is designed so that both the *Neighborhood* and the *Stencil* classes are inherited from a base class. The interface of this class provides generic operations on either of the derived classes – several of these operations are used in inter-process communication and boundary treatments. For example, one of the virtual member functions in the abstract base class is the *getBoundingBox* function. This function returns an abstract object, containing the extents of the *bounding box* described above. It gives a gross specification of the geometric region which forms the stencil/neighborhood. More details on how this can be used in inter-process communications are given in the next section.

4. PARALLEL IMPLEMENTATION

Our image processing toolkit is targeted toward Massively Parallel Processors (MPPs) or clusters of Symmetric Multi-Processors (SMPs).[11] On these architectures, communication between processors is done through the use of the Message Passing Interface (MPI) and the OpenMP libraries.[12,13] Several important issues have to be considered in order to design and implement an efficient parallel image processing toolkit. Many of these can be characterized as cost related.

Minimizing the cost of communication is critical to parallel performance and scalability of any software. In the MPI programming paradigm, data is communicated between processors as conceptual "sends" and "receives." The implementation of this send/receive mechanism is architecture dependent; but, as a rule, it is more expensive to carry out communication of arithmetic data than computation with the same data.[14,15] Another important issue is to minimize the time spent in first developing, and later, debugging, parallel algorithms. In light of these issues, our design approach seeks to:

- perform the communication efficiently in order to minimize its effect
- reduce the development and maintenance time through the re-use of common communication-related elements

To achieve these goals and incorporate flexibility into our software, it is desirable that the image processing operations be independent of the data distribution and communication paradigms. In other words, we want our algorithms to work regardless of how the user has configured the processors. To accomplish this, we need to incorporate the following into our design methodology[16]:

- develop data partitions and processor configurations
- determine the communication requirements based on the partition
- efficiently agglomerate the work
- map the work to the processors
- perform the actual work

For the stencil- and neighborhood-based operations mentioned in the previous section, many of the ideas for effectively implementing the above methodology have been studied extensively.[17–19] In particular, we can benefit from the work done in the fields of parallel numerical techniques for Linear Algebra and the solution of Partial Differential Equations. We exploit the fact that in general, the stencil/neighborhood operations have the following characteristics:

- *Local* – each task communicates with a small set of other tasks
- *Structured* – a task and its neighbors form a regular structure
- *Static* – the identity of communication partners does not change over time
- *Synchronous* – producers and consumers execute in a coordinated fashion, with producer/consumer pairs co-operating in data transfer operations

An effective way to address such problems is to first partition the image into contiguous rectilinear collections of pixels called *boxes*, and then to configure the processors to the resulting rectilinear partitioning.[20–22] A *box* specifies the lower and upper indices that denote the corners of a sub-image. The idea stems from an abstract "index space" associated with the original image. Note that the box object itself is small – it contains no actual data but only the indices representing the lower and upper corners which the sub-image occupies in the index space. It is included with the actual pixel data in a higher level object that represents the sub-image. As an example, consider a 2 dimensional M by N image. An abstract box associated with a sub-image $\{(i_0, j_0), (i_1, j_1)\}$ consists of indices (i, j):

$$(i,j) : 0 \leq i_0 \leq i \leq i_1 < M \quad , \quad 0 \leq j_0 \leq j \leq j_1 < N.$$

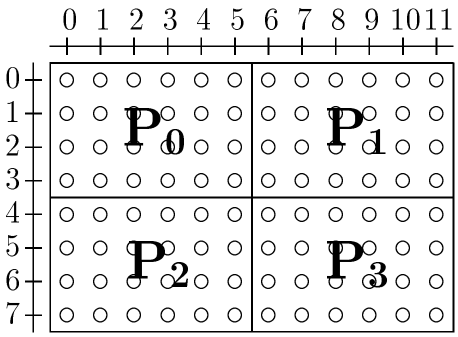

Figure 3. An 8 by 12 image decomposed on 2×2 processors

Figure 3 depicts an 8 by 12 "image" decomposed onto a 2×2 processor configuration with processors $\mathbf{P_0}$ through $\mathbf{P_3}$. If the image indices are from the region of index space $\{(0,0), (7,11)\}$, then the four boxes associated with each sub-image and processor assignment are:

- $\mathbf{P_0}$: $\{(0,0),(3,5)\}$,
- $\mathbf{P_1}$: $\{(0,6),(3,11)\}$,
- $\mathbf{P_2}$: $\{(4,0),(7,5)\}$,
- $\mathbf{P_3}$: $\{(4,6),(7,11)\}$.

To accommodate this functionality, we have included classes for "boxes" in our toolkit. These classes include functions that perform set or algebraic operations on boxes, often called *box calculus*.[20,22] Some of these operations include:

- grow/shrink - increase (or decrease) the size of a box in various ways
- refine/coarsen - refine (or coarsen) the index space associated with the box
- intersect/union - perform set manipulations with the index spaces

This box concept, along with the conventions adopted in MPI, enables us to address, directly or indirectly, the design methodology concepts mentioned earlier. An image processing application first uses MPI to create logically rectangular processor configurations and then maps the data onto the processor configuration in the form of boxes. To address performance concerns, our toolkit includes data distribution algorithms that partition the data and configure the processors so that the surface to volume ratio of the boxes is minimized. For operations based on regular grids, such as images, this minimization of the surface to volume ratio tends to minimize the cost of communication and maximize the performance.[23]

The stencil/neighborhood concept introduced earlier can be used to "package" general communication procedures within the toolkit. Since a stencil/neighborhood object contains information on the indices needed for local computations, the "gross" data requirements can be given for any such calculation in the form of a bounding box of the stencil/neighborhood operator. For instance, the 3 by 3 *Neighborhood* in Figure 2, has its bounding box described using the abstract box $\{(-1,-1),(1,1)\}$. The application of many linear and non-linear filters can use the stencil/neighborhood concept in order to create re-usable code. For example, if we ignore boundary treatment, a simple linear operation which averages the values of $a_{i,j}$ in a 3×3 area around a specific index (i,j) can be written as:

$$b_{i,j} = \sum_{n=-1}^{n=1} \sum_{m=-1}^{m=1} \frac{1}{9} a_{i+m,j+n}. \tag{1}$$

The same result can be achieved by creating a *Stencil* as the following nine index/value pairs:

$$\begin{array}{ccccc} \{(-1,-1);1/9\} & , & \{(0,-1);1/9\} & , & \{(1,-1);1/9\}, \\ \{(-1,0);1/9\} & , & \{(0,0);1/9\} & , & \{(1,0);1/9\}, \\ \{(-1,1);1/9\} & , & \{(0,1);1/9\} & , & \{(1,1);1/9\}, \end{array}$$

and using the following algorithm to "apply" the stencil to an image:

Algorithm: Apply Stencil

For each pixel location in the input image A (referred to as the input pixel)

- Initialize a temporary variable to zero
- For each element of the *Stencil* object
  - Get an offset/value pair from the *Stencil*
  - Multiply the stencil value with the image pixel obtained by adding the offset to the input pixel location and add the result to the temporary
- Assign the temporary to the output image B (at the same location as the input pixel)

With these abstractions, it is possible to create communication objects that accomplish all the communication associated with the application of a filter to an image in a parallel environment. One function of this object would be to take a processor configuration, along with the distributed image, and a given filter (with an associated stencil/neighborhood), and create a list of all sends and receives that must take place to permit local application of the filter. This would allow all interprocessor boundary exchanges associated with the application of a filter to be portable across processor configurations and data distributions. We next describe how this can be done in practice.

Each processor that owns a sub-image (that is part of a larger distributed image) can determine the regions of its sub-image that must be sent to another processor as follows:

Algorithm: Identify Send Regions

For all other processors which contain boxes of the distributed image

- Grow each processor's local box by the bounding box for the stencil/neighborhood to get a *destination box* with local plus "ghost" indices
- Intersect this *destination box* with the current processor's local box to obtain an *intersection box* with indices which need communicating
- Put the non-empty *intersection boxes* and corresponding processor ID on a list for communication

In the algorithm, the ghost indices refer to the the additional indices that result from the growth of each processors bounding box. In a similar way, each processor can determine the regions of other sub-images that it must receive as follows:

Algorithm: Identify Receive Regions

- Grow the current processor's box by the bounding box for the stencil/neighborhood to get a *source box* with local plus "ghost" indices
- For all other processors which contain boxes of the distributed image
  - Intersect the *source box* with the given processor's local box to obtain an *intersection box* with indices which need communicating
  - Put the non-empty *intersection boxes* and corresponding processor IDs on a list for communication

As an example, given the previously mentioned 3×3 neighborhood and a local 4×5 box, the send communication boxes are depicted as the 8 detached boxes in Figure 4, while the receive communication boxes are depicted as the 8 detached boxes in Figure 5.

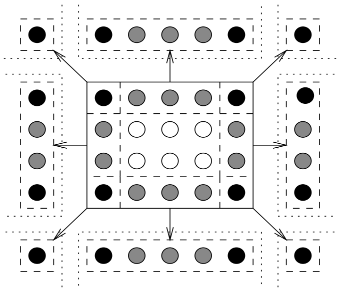

Figure 4. Send boxes for a 4 by 5 subimage

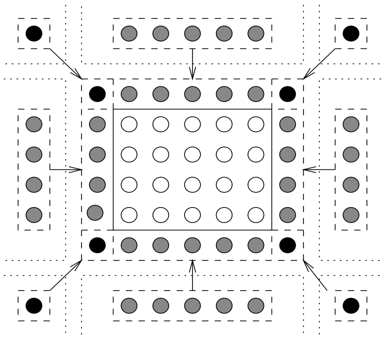

Figure 5. Receive boxes for a 4 by 5 subimage

This *stencil* and *box* idea can be used to implement the wavelet classes used in the denoising of image data, Section 5. Since a wavelet can be considered as a filter bank of high and low pass filters, a new wavelet category can be added by simply creating the filters as *stencils* and defining the wavelet coefficients appropriately.

In order to drive the image processing application, we let the user specify the distribution of the files to the processors as well as the processor topology. For example, the following input file

```
Name of File     Processors Used   Proc-Grid-Dimension    Proc-Topology

   file1           0 1 2 3              1D                    Linear
   file2           4 5 6 7              2D             Linear-X, Periodic-Y
```

indicates that the two input files, file1 and file2 are to be assigned to the group of processors $\{0, 1, 2, 3\}$ and $\{4, 5, 6, 7\}$, respectively. The four processors that operate on file1 are "connected" as a one-dimensional linear grid. The four processors that operate on file2 are connected as a two-dimensional grid, which is linear in the X-direction, but periodic in Y.

The preceding discussion applies to inter-processor boundaries and not the actual physical image boundaries. However, different boundary treatment methodologies can be incorporated into our parallel implementation paradigm. If a boundary treatment calls for mapping existing data into the boundary in some fashion (such as with periodic or reflecting boundaries), the corresponding mapped box and actual physical processor can be computed and handled in the intersection phase of the process. In the case where boundary treatment calls for numerical extension into the boundary (such as with extrapolation from the physical boundary), the actual extension can be handled separately from the above process. Finally, in the case where no boundary exchanges are incorporated, the above process could be modified to associate a specific stencil/neighborhood operation with an associated partition of the original image encompassing the boundary (i.e. a specific box). We have implemented the following boundary conditions in our wavelet classes:

- Periodic
- Whole point symmetry
- Half point symmetry
- Constant extrapolation from the boundary
- Extent with zero

The implementation of the boundary conditions is re-usable in the sense that once a stencil is given, the regions of boundary communication/computation can be constructed in a fashion similar to the interprocess communication regions above.

5. DENOISING IMAGE DATA USING WAVELETS

Denoising data by thresholding of the wavelet coefficients has been simultaneously proposed by several researchers during the past two decades. The method consists of applying a discrete wavelet transform to the original data, thresholding the detail wavelet coefficients, then inverse transforming the thresholded coefficients to obtain the denoised data.[24,25] There are several ways of calculating and applying thresholds.

The simplest threshold is the *Universal*,[26] $\sigma\sqrt{2\log n}$, where n is the sample size, and σ^2 is the noise variance. Threshold selection alternatives, based on minimizing certain optimization criteria, include the *minimax*,[26] and the *SURE*[27] methods. Thresholds can also be based on *hypothesis testing*, *cross-validation*, and *Bayesian* estimation approaches.[24] The most flexible of the threshold calculation methods, the *Top* method, involves selecting the threshold as a quantile of the empirical distribution of the wavelet coefficients. By experimenting with different quantile values, the user can interactively explore the best threshold for a given application.

The parallel implementation of the universal threshold is trivial, if the noise variance is known. Otherwise, it involves calculating certain measures of variability, like the standard deviation, L_p norm, or median absolute deviation (MAD),[28] in parallel. Calculating top thresholds in parallel requires a parallel sorting algorithm. Implementing some of the other threshold selection procedures, e.g. the minimax, in parallel requires optimizing certain risk functions in parallel.

Threshold, or shrinkage, application functions include the hard, the soft, and the semisoft functions.[29] The hard function involves a keep or kill strategy: coefficients whose absolute values are below a positive threshold are all "killed" (set to zero), while the others are kept unchanged. The soft function is similar to the hard, except that it either shrinks or kills: the coefficients that are kept are modified by shrinking them towards zero. The semisoft function generalizes the hard and the soft functions by using two thresholds, and includes both the hard and the soft as special cases.[30] Applying the thresholds in parallel is trivial in most cases, as once the threshold is selected, it is applied one-coefficient at a time. We note however, that certain Bayesian denoising schemes[24] do not involve separate

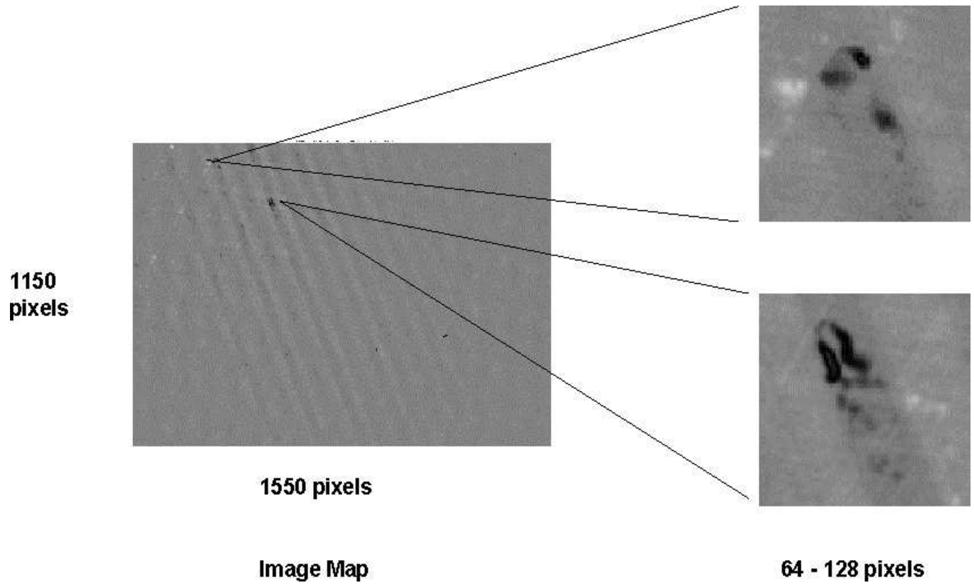

Figure 6. An example of a FIRST image map with two bent-double galaxies

threshold calculation and threshold application steps; rather, they shrink each wavelet coefficient by multiplying it by a function depending on the Bayesian parameters and on the coefficient itself.

The combination of soft shrinkage with the universal threshold is referred to as *VisuShrink*,[26] soft with the minimax threshold is called *RiskShrink*,[26] and soft along with the SURE threshold is known as *SureShrink*.[27] More recent advances, *BayesShrink*,[31,32] advocating soft shrinkage in a certain Bayesian framework, claim to outperform *SureShrink* estimates in the context of denoising images.

6. EXPERIMENTAL RESULTS

In this section, we describe some preliminary results of our work in progress in the area of denoising image data in parallel. The images we consider arise from the Faint Images of the Radio Sky at Twenty-cm (FIRST) survey.[33] Radio sources exhibit a wide range of morphological types. Of particular interest are sources with a bent-double morphology, as they indicate the presence of large clusters of galaxies. Currently, FIRST scientists identify bent-doubles by manually looking through the images. The image dataset is "only" about 200 Gigabytes, moderate by today's emerging standards, but large enough to inhibit an exhaustive visual inspection by the astronomers. Our goal is to automate the detection of bent-doubles by using data mining techniques.[34]

The FIRST survey is producing the radio equivalent of the Palomar Observatory Sky Survey. Using the Very Large Array (VLA) at the National Radio Astronomy Observatory (NRAO), FIRST is scheduled to cover more than 10,000 square degrees of the northern and southern galactic caps, to a flux density limit of 1.0 mJy (milli-Jansky). At present, with the data from the 1993 through 1998 observations, FIRST has covered about 6,000 square degrees, producing more than 20,000 image maps. Each image map is 1550 by 1150 pixels, large enough to benefit from parallel processing. Due to the sensors used to collect the data, there is a pronounced noise pattern that appears as "streaks" in the image. Figure 6 shows an image map that contains two bent-double galaxies.

The task of denoising these images is made challenging by the fact that some of the information necessary to identify a galaxy as a bent-double, could lie on a "streak" and be removed as "noise". To ensure that wavelet denoising can indeed be applied to our images without any significant loss of useful information, we first experimented with the serial version of our code on a small image extracted from the larger image map.

Figure 7 presents an example FIRST image, and various wavelet denoised versions of it. All the examples were obtained using the Haar wavelet, and three multiresolution levels in the wavelet decomposition. We are currently experimenting with other wavelets and denoising options to find an optimal combination for the FIRST dataset. A good technique should remove the background noise effectively, but keep the important bent-double features intact. For example, in the bent-double of Figure 7, the two wavy lobes of the bent-double are connected by a fainter bridge. This bridge is important as it can be used to calculate the "angle" of the galaxy to determine if it is a bent-double or not. As the bridge lies on one of the noise streaks, we have to be careful in the use of denoising. This task is made more challenging by the fact that the number of radio galaxies precludes individual examination of the effects of denoising on each image.

7. SUMMARY AND CONCLUSIONS

In this paper, we have described our work on the design and implementation of an object-oriented parallel toolkit for image processing. Using examples, we have illustrated how the object-oriented paradigm can help us to abstract out the commonalities across several different operations. This enables re-use of software, not only in the implementation of the image processing operations, but also the communication tasks that must be supported for parallel implementation. We showed how ideas developed in the fields of partial differential equations and linear algebra can be exploited to make our software portable across data distributions and processor configurations. We present some preliminary results that show that wavelet denoising techniques can be used effectively on smaller images, and, through the use of our parallel software, on larger images as well.

ACKNOWLEDGMENTS

UCRL-JC-138953: This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

REFERENCES

1. "Sapphire: Large-scale Data Mining and Pattern Recognition." *http://www.llnl.gov/casc/sapphire.*
2. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining*, MIT Press, Cambridge, Mass., 1996.
3. U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "The KDD Process for Extracting Useful Knowledge from Volumes of Data," *Communications of the ACM Special Issue on Data Mining* **39**, pp. 27–34, 1996.
4. N. Ramakrishnan and A. Y. Grama, "Data Mining: From Serendipity to Science," *IEEE Computer Special Issue on Data Mining* **32**(8), pp. 34–37, 1999.
5. Jähne, B., *Practical Handbook on Image Processing for Scientific Applications*, CRC Press, Boca Raton, Florida, 1997.
6. A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.
7. J.-L. Starck, F. Murtagh, and A. Bijaoui, *Image Processing and Data Analysis: The Multiscale Approach*, Cambridge University Press, Cambridge, United Kingdom, 1998.
8. E. Arge, A. Bruaset, and H. P. Lantangen, "Object oriented numerics," in *Numerical Methods and Software Tools in Industrial Mathematics*, Dæhlen, M. and Tveito, A., ed., pp. 7–26, Birkhäuser, 1997.
9. "Flexible Image Transport System (FITS)." *http://www.cv.nrao.edu/fits/FITS.html.*
10. I. Daubechies, *Ten Lectures on Wavelets*, SIAM, 1992.
11. D. E. Culler and J. P. Singh, *Parallel Computer Architectures A Hardware/Software Approach*, Morgan Kaufmann, 1999.
12. University of Tennessee, *MPI: The Message Passing Interface Standard*, 1.1 ed., 1995.
13. "OpenMP Application Program Interface." *http://www.openmp.org.*
14. J. D. McCalpin, "A survey of memory bandwidth and machine balance in current high performance computers," tech. rep., Silicon Graphics Computer Systems, 1997.
15. K. Dowd and C. Severance, *High Performance Computing, Second Edition*, O'Reilly and Associates, 1998.
16. I. Foster, *Designing and Building Parallel Programs*, Addison-Wesley, 1995.
17. S. F. Ashby, R. D. Falgout, T. W. Fogwell, and A. F. B. Tompson, "Numerical simulation of groundwater flow on mpps," in *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra*, 1993.
18. V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction To Parallel Computing: Design And Analysis Of Algorithms*, Benjamin-Cummings / Addison-Wesley, 1994.

(a) (b) (c) (d) (e) (f) (g) (h)

Figure 7. Examples of original and denoised FIRST image maps. (a) Original. (b) Top 0.1st and 0.9th quantile thresholds; semisoft. (c) Universal, σ as the L_3 norm of the level_1 detail coefficients; hard. (d) As (c); soft. (e) Universal, σ as the L_1 norm of all the detail coefficients; hard. (f) As (e); soft. (g) Universal, σ as the MAD of the level_1 detail coefficients; hard. (h) As (g); soft.

19. G. C. Fox, R. D. Williams, and P. C. Messina, *Parallel Computing Works!*, Morgan Kaufmann Publishers, San Francisco, 1994.
20. W. Y. Crutchfield and M. L. Welcome, "Object oriented implementation of adaptive mesh refinement algorithms," *Scientific Programming* **2**, pp. 145–156, 1993.
21. R. D. Falgout and J. E. Jones, "Multigrid on massively parallel architectures," Tech. Rep. UCRL-JC-133948, Lawrence Livermore National Laboratory, 1999.
22. "Samrai : Structured adaptive mesh refinement applications infrastructure." http://www.llnl.gov/casc/SAMRAI/.
23. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors*, Prentice Hall, 1988.
24. R. T. Ogden, *Essential Wavelets for Statistical Applications and Data Analysis*, Birkhäuser Boston, 1997.
25. B. Vidakovic, *Statistical Modeling by Wavelets*, Wiley Series in Probability and Statistics, John Wiley & Sons, Inc., 1999.
26. D. L. Donoho and I. M. Johnstone, "Ideal spatial adaptation via wavelet shrinkage," *Biometrika* **81**, pp. 425–455, 1994.
27. D. L. Donoho and I. M. Johnstone, "Adapting to unkown smoothness via wavelet shrinkage," *Journal of the American Statistical Association* **90**, pp. 1200–1224, December 1995.
28. W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S-Plus*, Springer, 1996.
29. A. Bruce and H. Gao, "Understanding waveshrink: Variance and bias estimation," Tech. Rep. 36, StatSci Division of MathSoft, Inc., 1995.
30. H. Gao and A. Bruce, "Waveshrink with semisoft shrinkage," Tech. Rep. 39, StatSci Division of MathSoft, Inc., 1995.
31. G. Chang, B. Yu, and M. Vetterli, "Spatially adaptive wavelet thresholding based on contect modeling for image denoising," *IEEE Trans. Image Processing* , 1998. Accepted for publication.
32. G. Chang, B. Yu, and M. Vetterli, "Adaptive wavelet thresholding for image denoising and compression," *IEEE Trans. Image Processing* , 1998. Accepted for publication.
33. R. H. Becker, R. L. White, and D. J. Helfand, "The FIRST Survey: Faint Images of the Radio Sky at Twenty-cm," *Astrophysical Journal* **450**, p. 559, 1995.
34. I. K. Fodor, E. Cantú-Paz, C. Kamath, and N. A. Tang, "Finding bent-double radio galaxies: A case study in data mining," in *Computing Science and Statistics*, vol. 33, 2000.