Data Parallel Programming with the Khoros Data Services Library

Steve Kubica, Thomas Robey, Chris Moorman

Khoral Research, Inc.
6200 Indian School Rd. NE Suite 200
Albuquerque, NM 87110 USA
E-mail: info@khoral.com

1 Introduction

Khoros is a powerful, integrated system which allows users to perform a variety of tasks related to image and signal processing, data exploration, and scientific visualization. Khoros includes a visual programming language, interactive image display and plotting programs, and an extensive set of image processing, data manipulation, scientific visualization, geometry and matrix operators. Additionally, Khoros contains a suite of software development tools that allow users to create new applications[1].

Cantata, the visual programming environment available with Khoros, provides a tool for algorithm development and rapid prototyping. Users can construct visual programs within Cantata by connecting glyphs, which represent the actual data processing or visualization operators to be executed, with data connections, which represent the data flow between these operators. This intuitive approach provides a simple and accessible way to interconnect and execute many different programs.

2 Khoros Support for High Performance Computing

DARPA, Wright Laboratories, and Phillips Laboratories, in conjunction with Khoral Research, have developed a combined plan to extend the Khoros framework to support parallel computers and eventually even embedded High Performance Computers (HPC)[2]. This Advanced Khoros system will provide support for the development and creation of high-performance parallel and embedded systems using a common framework for all levels of development.

Research in parallel algorithms has tended to focus on speeding up complex algorithms involving complex communication patterns. The focus of Advanced Khoros is to handle the simpler parallelization common in many image and signal processing applications and to develop programming tools that simplify the time-consuming conversion of serial to parallel programs.

2.1 Task Parallelism

Task Parallelism, or course-grained parallelism, refers to the ability to run multiple separate processes in parallel. Multiple branches in a workspace are able to run simultaneously. With task parallelism, there is no interprocess communication between the workspace operators beyond their input and output connections.

2.2 Data Parallelism

Data Parallelism, or fine-grained parallelism, refers to the ability to process data in parallel within a single routine. Data parallelism would be exhibited by a single Khoros operator executed in a Single Program, Multiple Data (SPMD) manner processing a scattered data set across a number of processors. This data parallel operator would distribute or scatter an input data set across multiple processes, process the data, and potentially regather the output.

3 A Merging of Paradigms

There are two paradigms to consider in writing parallel routines:

– *The message passing programming paradigm* is exemplified by the Message Passing Interface (MPI). Data distribution is explicitly handled by the programmer using direct MPI communication calls. This is flexible, but not convenient. All communication must be explicitly managed by the programmer. Developers who use MPI calls directly must deal with a fair amount of low- level complexity when structuring the intercommunication between parallel processes. This complexity can be prohibitive for developers with little or no parallel programming experience who wish either to write new parallel code or to migrate existing serial code to a parallel machine.

– *The data parallel programming paradigm* is exemplified by High Performance Fortran (HPF). In HPF, a data distribution is specified on an array with a compiler directive; the compiled code will automatically handle the data distribution. While this mechanism is convenient, the compiler often does not choose optimal strategies, and there is no mechanism for optimization.

Distributed data services provides the easy-to-use features of HPF and the flexibility of MPI. Data distribution will be handled automatically, reducing the work to create parallel programs, while low-level communication calls can still be used as needed as for critical code speed-up.

4 Data Services Overview

Data services provides access to data through an abstract data object via domain-specific data models. The polymorphic data model is the most commonly used, and is accessible through the polymorphic data services library. The polymorphic data model provides a uniform interpretation of data across many different processing domains. Consisting of a number of data segments representing a time-series of volumes in space, the structure of this model is ideal for storing signal, image, and volumetric data. The primary data segment, known as the value segment, is used to store the actual values intended for processing[3].

The data services application programming interface (API) consists of a set of library functions that operate on an abstract data object. The internals of the data object are kept hidden from the developer; the data contained in the object can never be accessed directly. Access is only possible through the data service library calls. Within a program, a data object is declared to be an instance of the abstract data type kobject. Every data processing routine in Khoros uses these objects in approximately the same manner; data is read in from one or more input objects, processed, and then written out to one or more output objects.

Data objects contain primitives and attributes. Primitives are used to access data within the data object and attributes are used to access meta-data within the data object. Meta-data is a term used loosely to cover characteristics of the data such as size and data type as well as auxiliary information such as the date or a comment. User-defined attributes can also be created to store any needed application-specific information.

4.1 Distributed Data Services

Distributed data services is implemented as an extension to polymorphic data services. These extensions allow a developer to distribute the value segment across several different processors for processing. Following the SPMD model, each processor will be running the same program, but will be operating on a different subset of the overall value segment. The scattering of data to each processor and the gathering of the final results will be handled within data services. Since the only modifications in the API are made for handling this data distribution, developers should be able to easily migrate their existing polymorphic data services code to run on parallel machines.

The key idea behind distributed data services is the concept of a distributed data object. A distributed data object is an object that consists of many subobjects distributed over several different processors. When viewed collectively, these subobjects act as local handles to a conceptual global object. Each processor only has access to the portion of the value segment which is contained in its local subobject. The global characteristics are accessible to each processor from each local subobject via special attributes. With a few exceptions, distributed data objects are handled just like regular data objects in terms of accessing data and attributes.

4.2 Process Groups and Communicators

The concept of groups and communicators has been borrowed from MPI. In MPI, the set of processes involved in an interprocess communication is defined to be a group. Each process within a group is given a numerical rank, identifying it uniquely within that group. For a given communication call, a communicator handle is used to identify the group in which that communication will occur. `MPI_COMM_WORLD` is the communicator for the process group which contains all processes; that is all the processes that were collectively initiated with a call to `MPI_Init`. New communicators can be created as subsets of this communicator. This provides a mechanism for isolating the communication to subgroups within the group of all processes. For distributed data services, a communicator is used to specify over which processes data distribution shall occur.

The handle `KCOMM_WORLD` is initialized to refer to the communication group that contains all processes. Khoros extends the concept of groups or communicators by using scattered and gathered states for a communicator where gathered implies all data resides on the root node. Within each communication group, the process with a rank of 0 is considered to be the root process for that group. For any given Khoros communicator, the communication group that contains only the root process is given by the function `KCOMM_SELF()`.

4.3 Shape Description

The shape of the data object refers to the manner in which a dataset is distributed across a series of processors. The shape is defined in terms of the polymorphic data model, and dictates how each segment in the model is divided over a process group. The polymorphic value segment consists of a five-dimensional array of data; it is best pictured as a time-series of volumes, where every element of the volume is a vector of data values. Partitioning is always along a single dimension of the polymorphic data model.

The shape is controlled via a data services attribute: `KPDS_DISTRIBUTION`. The effect of setting this attribute is immediate; the data object will take on the new shape when the set attribute call is made. Setting this attribute is done collectively by all processors on their local kobject handles. All processors must set the same attribute values.

The `KPDS_DISTRIBUTION` attribute consists of three values:

- `DIMENSION`
  This value describes the polymorphic dimension along which the partitioning will occur. Eventually, partitioning in more than one dimension may be achieved using recursion.

- DISTRIBUTION
  This value describes how the data is to be distributed along the distribution dimension. Possible values are KREPLICATED, KBLOCK, or KCYCLIC. A KREPLICATED distribution implies that the entire dataset is replicated to each processor. A KBLOCK distribution implies that the overall size of the partition dimension is divided by the number of processors, with each processor getting a contiguous block of data along that dimension. A KCYCLIC distribution implies that every slice of the partition is distributed to a different processor in a cyclic manner.

- COMMUNICATOR
  This value specifies the Khoros communicator over which the distribution should be performed. The value KCOMM_WORLD will dictate that the data should be distributed over all processors. The value KCOMM_SELF(KCOMM_WORLD) will distribute the data only to the root process. Note that data distributed to one process is effectively gathered on that process; hence changing the communicator value is used to scatter and gather the data.

Any existing data file can be opened as a distributed data object and will have a communicator of KCOMM_SELF(KCOMM_WORLD). A scatter can be accomplished simply by setting a communicator value of KCOMM_WORLD on a distributed data object.

Since many signal and image processing algorithms require data from adjacent blocks when processing, a KPDS_OVERLAP attribute is also provided to specify an overlap for KBLOCK distributions. These overlaps affect the size of each local subobject on the distributed dimension; the first and last subobjects are expanded by the overlap size while the other processors are expanded by two times the overlap size. When collecting distributed overlapped data, only the non-overlapped regions from each subobject will be collected. This attribute has no affect on KCYCLIC or KREPLICATED distributions.

4.4 Related Work

This specific concept of distributing polymorphic data objects is not completely new. Very similar work on a parallel Khoros Data Services was done by Doug Warner at the Albuquerque Resource Center (ARC). The main difference between Warner's approach and that presented here is that his required an explicit gathering glyph to recollect the data[4]. A similar concept has also been investigated by Nathan Doss and Thomas McMahon at Mississippi State University[5]. In their approach, a polymorphic data object is explicitly distributed which produces local data objects for each machine. The glyph accepting input will block on the opening of the input until the glyph producing output executes a close call for the output. At that time, the communication between the two glyphs, defined to be in different MPI groups, will take place. This communication will be entirely in memory. This approach is particularly interesting since both routines are running simultaneously during the communication. One way in which our approach differs from theirs is that each processor is allowed access not only to the local data object, but also to the initial global data object. Their approach requires multiple copies of the data, contained in multiple subobjects.

5 Summary

The distributed data service library allows developers to control the distribution of data in a parallel program simply by setting attributes on a distributed data object. This interface provides the power of a data parallel programming paradigm by abstracting all the low-level communication required for effecting a data distribution. The simplicity of the interface should also facilitate the porting of serial routines to run on parallel architectures.

The emphasis in the development of distributed data services has been to provide a framework which addresses the common problems encountered in writing even the simplest data parallel programs, while still being extensible to problems which are less frequently encountered, and often more complicated. Distributed data services allows developers writing such applications to use low-level MPI calls as necessary. In this way, distributed data services combines the convenience of a data parallel language such as HPF with the flexibility of the message passing library MPI.

AVIS Contract Acknowledgment
This project is supported by Federal Contract Number N660001-96-C-8619 sponsored by
DARPA/ITO
3701 N. Fairfax Drive
Arlington, Virginia
22203-1714

MAUI Contract Acknowledgment
This project is supported by Federal Contract Number F296001-96-C0006 sponsored by
United States Air Force
Air Force Systems Command
Phillips Laboratory (PL)
Kirtland AFB, New Mexico
87117-6008

References

1. John Rasure and Steve Kubica. The khoros application development environment. In H.L. Christensen and J.L. Crowley, editors, *Experimental Environments for Computer Vision and Image Processing.* World Scientific, 1994.
2. Khoral Research Inc and the DARPA/WL Khoros Coordinating Committee. Advanced khoros software development plan. White paper produced as deliverable for KRI contract to DARPA/WL Advanced Khoros Working Group, 1996.
3. Khoral Research, Inc. *Khoros Manuals: Programming Services Volume II*, 1994.
4. Doug Warner. Parallel khoros data services. In *Parallel Khoros Workshop.* Albuquerque Resource Center, April 1996.
5. Nathan Doss and Thomas McMahon. An integrated khoros and mpi system for the development of portable dsp applications. Department of Computer Science and NSF Engineering Research Center for Computiation Field Simulation, Mississippi State University, 1996.

This article was processed using the LaTeX macro package with LLNCS style