CERSe - a Tool for High Performance Remote Sensing Application Development

Nathan A. DeBardeleben
864-656-7223
ndebard@parl.clemson.edu

Walter B. Ligon III
864-656-1224
walt@parl.clemson.edu

Sourabh Pandit
864-656-7223
spandit@parl.clemson.edu

Dan C. Stanzione Jr.
864-656-7367
dstanzi@parl.clemson.edu

Parallel Architecture Research Lab
Department of Electrical and Computer Engineering
Clemson University
105 Riggs Hall
Clemson, SC 29634-0915
http://www.parl.clemson.edu

Abstract— As the quality and accuracy of remote sensing instruments available improves, the ability to quickly process remotely sensed data is in increasing demand. A side effect of this increased quality and accuracy is a tremendous increase in the amount of computational power required by modern remote sensing applications.

In this work, a problem solving environment for parallel computing named the Component-based Environment for Remote Sensing (CERSe) is described. CERSe is targeted at producing applications for Linux Beowulf clusters. CERSe uses out-of-core computation techniques to handle extremely large datasets. A multi-threaded, multi-queue runtime engine executes user-supplied modules in parallel over partitioned datasets. The modules supplied by the user can be purely sequential code. Parallelism comes from the partitioning of the dataset by the system and from the ability to execute multiple modules simultaneously. Parallel I/O is used to take advantage of the I/O resources throughout the cluster and provide high performance. A GUI is provided which allows users to specify a dataflow for the application by creating and connecting pluggable modules. The GUI also provides facilities for launching jobs, selecting parameters and data, and analyzing job performance and results.

CERSe's runtime engine and graphical user interface are presented. Performance results with and without use of a parallel file system are given which show near linear speedup can be achieved through I/O tuning.

Table of Contents

1. Introduction

There is a growing number of people who want to use remotely sensed satellite data and Geographic Information Systems (GIS) data. The different applications that users want to run require increasing amounts of temporal, spectral, and spatial resolution. Some users, for instance, are satisfied with a single image a day, while others require many images an hour. Scientists are beginning to take advantage of multi-spectral datasets obtained from satellites which, in many cases, have over 100 different spectral bands present. Also, there is growing interest in combining large numbers of datasets from different satellites taken at different times of different geographical locations and at different spotsize resolutions [1].

For users to take advantage of this data they must be provided with software tools. These tools must deal with the issues inherent in the growing data requirements. How to store and how to process the data are two such issues.

The problems facing the remote sensing community are:

- a diversifying user base requires lower cost solutions,

- increasing amounts of temporal, spectral, and spatial resolution requires massive data storage and processing power (parallel computers),
- parallel computers are difficult to program,
- rapid code development, execution, analysis, and maintenance remains of major importance.

Beowulf clusters [2] are large computers comprised of commodity off-the-shelf (COTS) hardware components such as cheap memory, EIDE disks, inexpensive networks, and conventional CPUs. The software (including operating system) is low cost / no cost and parallel programming libraries and tools are available for development of user code. This hardware organization lends itself well to coarse-grain problems. Coarse-grain problems are those which have portions that can operate a relatively long time without necessary communication to other portions. Beowulf computers also show a beneficial cost-to-performance ratio over conventional supercomputers. For these reasons, our approach targets Beowulf clusters.

We propose the design of a problem solving environment (PSE) which meets the needs of the remote sensing community by providing high performance, ease of use, and extensibility. This environment must

- Provide a mechanism for code reuse,
- Run on parallel computers,
- Be usable without advanced knowledge of parallel architecture or parallel algorithms, and
- Provide tools which facilitate fast algorithm development and data analysis (such as a graphical environment, data selection tools, and performance analysis agents).

We discuss a problem solving environment aimed at achieving these goals. This system accomplishes the goals of extensibility and code reuse through the use of interconnected components, or modules, to design algorithms. These modules consist of reusable blocks of code which are run in parallel on a parallel computer. Issues dealing with how to achieve the parallelism, passing data between modules, and synchronization are hidden from the user. We provide tools to address usability such as graphical dataflow editors, data visualization, data selection tools, and tools to constrain the region of interest.

A number of related projects exist, some of which specifically target remote sensing applications while others target different domains. The Common Component Architecture [3] is a framework for defining the interaction between components as well as an interface definition language (SIDL). Khoros [4] is a commercial software package for modular program development on sequential computers where users describe the dataflow with “glyphs” which are separate programs. Cactus [5] is a software tool for parallel computers which defines the interaction between the driver (flesh) and modules (thorns) aimed primarily at astrophysics codes. The Open Source Software Image Map (OSSIM) [6] aims to leverage existing open source tools in the remote sensing community to construct an integrated tool for running remote sensing codes on parallel machines.

In the next section we outline the design of the problem solving environment framework upon which CERSe is built and discuss CERSe in detail. An example of porting an existing NASA remote sensing module into CERSe is given next. We conclude with performance results and discuss both computational and I/O speedup.

2. CERSe

The Component-based Environment for Remote Sensing (CERSe) is a software tool for developing modular codes for parallel computers. CERSe modules are subroutines in either C or Fortran. Modules are placed onto a canvas and a dataflow graph is created by connecting inputs and outputs between modules. This model works very well for many remote sensing applications. Once the dataflow graph has been created, CERSe jobs may be launched on a parallel computer (such as a Beowulf). Special, internal data structures are maintained which handle passing data between modules, allocating and deallocating memory when needed and communication between processors. Programmers have access to these data structures but most find that the helper functions for retrieving and creating data are enough. Advanced users can create modules which break satellite datasets into partitions, pass them to different processes on the parallel machine, and reassemble the results. Queues of data to be processed are maintained internally which facilitate load balancing between the processors. With these queues, once a processor is done with a portion of the dataset it merely requests an additional portion from the master process.

A graphical user interface (GUI) assists in application development by providing a place for describing the dataflow graph, launching jobs on the parallel computer, visualizing results in real-time, and analyzing performance after job completion.

CERSe is built upon two additional technologies - Coven and the Algorithm Description Format (ADF). ADF [7] is an open graph format which defines the interaction between connected modules. Coven is a low-level toolkit for building problem solving environments for parallel computers. Coven has been used to build PSEs for physics and electromagnetics as well as remote sensing.

Figure 1 depicts the relationship between ADF, Coven, and CERSe.

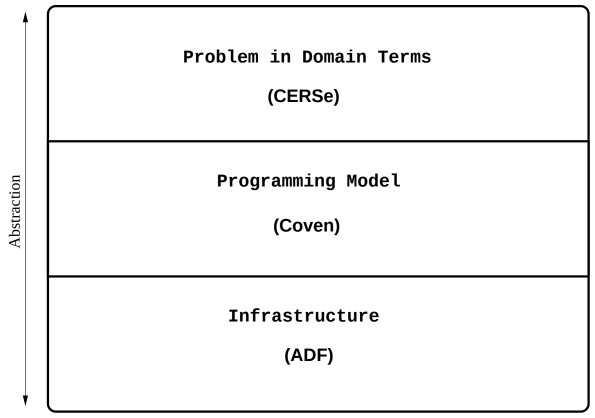

Figure 1: PSE Architecture

Coven PSE Framework

Coven provides a framework for developing PSEs for parallel computers. A *runtime driver* and *GUI client* can be extended and utilized by specific PSEs.

Coven's runtime driver is composed of:

- TPH - The Tagged Partition Handle is an internal data structure for representing data flow between modules.
- Module Loader - Modules are loaded dynamically by the module loader so that all operations are done in-core.
- Program Sequencer - The program sequencer runs on each processor of the parallel machine and transparently handles executing modules in order and passing the required data between them.
- Threading - Modules are placed into threads within processes so that additional parallelism can be achieved through asynchronous I/O, computation, and network communication.
- Queues - Input queues of TPHs are maintained for each thread on each processor which allows for load balancing.

Coven's GUI client provides:

- Graph Editor - The editor is a tool for graphically describing the dataflow between modules.
- Code Generator - Dataflow algorithms are transformed into code which is used by the Coven runtime driver by the code generator.
- Introspection - Coven parses user source code modules to determine the types of all inputs and outputs and provides tools for type checking.
- Performance Analysis - After jobs have been run, Coven provides performance analysis of many internal events which can be useful for hand-tuning.

CERSe customizes the Coven framework to the remote sensing domain by changing the terminology presented in the interface to be more familiar to the remote sensing user. Additionally, CERSe provides graphical agents for selecting datasets through a database, choosing a region of interest, and real-time visualization of results as they are computed on the parallel machine.

Achieving Parallelism

Parallelism is achieved with CERSe through the use of distributor modules which partition satellite datasets and distribute them to separate processes on the parallel machine. While the distributor modules do require some knowledge of parallel computing, many are provided which simply can be dropped into place. Advanced users can create their own. Functionality within Coven allows advanced users to use the Message Passing Interface (MPI) [8, 9] to communicate between parallel modules if a problem does not fit the simple data partitioning model. Currently, no users of CERSe have found the MPI functionality a necessity.

Additional performance gains come from asynchronous I/O, computation, and communication through the use of Coven's multi-threaded feature. The inherent queuing system also improves performance by creating a pipeline effect as data passes through the system from thread to thread.

3. CERSe Example

An example of a module is one that calculates the satellite zenith angle for each pixel in an AVHRR satellite image. This example shows the process of taking existing (legacy) code and incorporating it within CERSe.

Figure 2 depicts the module code to calculate the satellite zenith angle for each pixel over each scan line partitioned to the module.

The beginning of the `satzenith` function uses four calls to CERSe helper functions. These functions retrieve the requested data from the TPH transparently. In CERSe, attributes are single values (such as integers, strings, and floats) and buffers are arrays (possibly multi-dimensional). Several calls in the `satzenith` module are made to retrieve attribute values for the number of scan lines that this module will process and the number of points in a scan. This module will be run by many processors many times and each time will operate on only a portion of the data. The variable `num_scan_lines` determines each time the module runs exactly which scanlines will be processed.

The `CERSe_INPUT` function reads from the TPH the Brouwer Lydanne osculating elements and assigns the pointer to a local variable named `oscs`. The `CERSe_OUTPUT` function allocates an array of floating point values and assigns the pointer to the local name `satzens`. Some constant values pertaining to the maximum scan angle and the radius of the earth are also defined. Recognizing that the earth is not a sphere, as this code assumes, one could readily implement code which calculates the earth radius for particular scan lines.

Finally, there are two loops which iterate over the scan lines given to a particular processor. These loops are where the parallelism occurs since each processor operates on only a portion of the total number of scan lines. As this module is executed by the Program Sequencer on each slave node the TPH will contain data relevant only to a portion of a dataset. We achieve a great deal of data parallelism with this method. The satellite zenith angle is calculated for each point in these loops. These values are placed into the `satzens` data structure which was allocated by the call to `CERSe_OUTPUT` as an array of float values and is now present in the TPH. This array is now available in the TPH for other modules to use as input.

The code in Figure 2 came from existing NASA remote sensing tools.

4. Performance Results

For our performance tests we used CERSe to create a common remote sensing application. This application computes the Normalized Difference Vegetation Index [10], masks clouds, and projects the output to a common coordinate system. Sequential remote sensing code exists which performs these algorithms and was used in speedup analysis.

In an effort to distinguish speedup attributed to computation versus I/O, three sets of tests of CERSe were run on a dedicated cluster of computers. The cluster consisted of eight nodes connected by Fast Ethernet. Each node had 64MB of RAM, 80GB of disk space, and a 300MHz Pentium II processor. For the first series of tests no parallel file system was used. In these tests all the datasets consumed by CERSe were available on the local disks of each of the compute nodes in the cluster. Therefore, the data was replicated eight times. This kept all I/O accesses local and shows the computational speedup of CERSe without effects of I/O. Figure 3 shows the speedup curve for this simulation. It can be seen that using only local data provides very good speedup. This is to be expected as the problem is naturally parallel. A clear drawback of this simulation, however, is that it requires all the data to be replicated on all of the local disks within the cluster. With the massive amounts of data currently available and the increasing volume of new data this solution is not practical. To deal with this problem we incorporated parallel I/O.

Tests of CERSe were also run using PVFS [11, 12], a parallel file system for Beowulf clusters developed at the Parallel Architecture Research Laboratory at Clemson University. PVFS stripes data across the disks in a Beowulf cluster like a RAID. In the first set of tests all of the processors used performed both computation and I/O. Thus, when less than the maximum number of processors was used, the unused nodes were not used as I/O nodes. The datasets were partitioned along scanline

```
#include "CERSe.h"

double totalswath = 110.7968;
double maxscanangle = 55.37;
double earthRadius = 6378140.0;

CERSe_MODULE(satzenith, tph)
{
   int num_scan_lines = CERSe_INT_ATTRIB_GET(tph, "num_scan_lines", "Number␣of␣scanlines␣for␣THIS␣TPH");
   int points_per_scan = CERSe_INT_ATTRIB_GET(tph, "points_per_scan", "Number␣of␣points␣in␣a␣scanline");
   CERSe_OSCEL_BUFFER_TYPE oscs = CERSe_INPUT(tph, CERSe_OSCEL_BUFFER, "Brouwer␣Lydanne␣osculating␣elements");
   CERSe_FLOAT_BUFFER_TYPE satzens = CERSe_OUTPUT(tph, CERSe_FLOAT_BUFFER, num_scan_lines*points_per_scan, "Satellite␣zenith␣angles");

   int scan, i;
   float *latsptr, *lonsptr;
   double SemiMajorAxis, Eccentricity, MeanAnomaly, sat_height, ang, tmp;

   /* ‘scan’ is a relative scan, 0->number of scans instead of start_scan -> start_scan+number of scans */
   for(scan=0; scan<num_scan_lines; scan++) {
      SemiMajorAxis = oscs[scan].SemiMajorAxis;
      Eccentricity = oscs[scan].Eccentricity;
      MeanAnomaly = oscs[scan].MeanAnomaly;

      sat_height = (SemiMajorAxis * (1 - (Eccentricity * Eccentricity))) / (Eccentricity * cos(MeanAnomaly) + 1);

      /* loop over each angle in the scan line */
      for(i=0; i < points_per_scan; i++) {
         ang = fabs(totalswath * i / points_per_scan - maxscanangle);
         /* calculate the zenith angle */
         tmp = asin((sat_height / earthRadius) * sin(ang * RADDEG));
         tmp = fabs(tmp);
         satzens[(scan * points_per_scan) + i] = tmp;
      }
   }
}
```

Figure 2: Satellite Zenith Angle Calculation Module

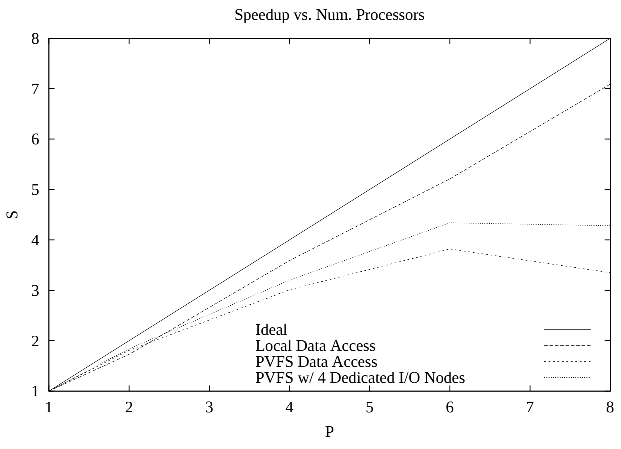

Figure 3: Speedup for Varying Number of Compute Nodes Using Local Data Access

boundaries and the default PVFS stripe size of 64KB was used initially. The second PVFS test case involved using four dedicated I/O nodes and varying numbers of compute nodes. Again, the default PVFS stripe size was used. These tests show the speedup due to I/O.

As can be seen from the plot, PVFS incurs considerable overhead, but exhibits reasonably good scalability. As the number of processors increases beyond six, however, we find that the speedup when using PVFS decreases due to the file system being overloaded. The tests using dedicated I/O nodes suggest that I/O tuning can have beneficial effects. We further believe that a balance exists between stripe size and the size of partitions that are distributed out to each processor. The configuration of the file system can have a significant impact on both overhead and scalability and we are currently working to improve the interaction between PVFS and CERSe.

5. Conclusions and Future Work

The remote sensing community faces a number of challenges which are addressed by CERSe. The problem solving environment presented allows for existing remote sensing code and new algorithms to be written in a modular framework which can be executed on a parallel computer. Advanced understanding of parallel computing has been abstracted away from the user. Parallel computing is essential for handling the increasing amounts of data due to growing temporal, spectral, and spatial resolutions.

CERSe is designed to provide high performance computing, extensibility, and usability for remote sensing problems. An environment for modular code development and easy pluggability of modules is provided. This environment makes it easy to both incorporate existing, legacy software as well as develop new algorithms. A graphical interface addresses usability by providing tools for dataflow graph creation, data visualization, and additional functionality specific to remote sensing problems such as specification of region of interest, choosing map projections, and database connectivity. Performance results show that near ideal computational speedup is achieved by CERSe. I/O tuning is still an issue and research is currently under way to address this.

Additional reserach has begun on code profiling within Coven to allow users to visualize time spent within modules, threads, and processes. Coven is being used to build environment for other realms of science such as computational fluid dynamics and material simulations.

Acknowledgments

This work was supported in part by NASA grant NAG5-8605 and by the ERC Program of the National Science Foundation under Award Number EEC-9731680.

References

[1] M. Halem, F. Shaffer, N. Palm, E. Salmon, S. Raghavar, and L. Kempster, “Can We Avoid A Data Survivability Crisis?,” Tech. Rep. 51, National Aeronautics and Space Administration, 1999.

[2] T. Sterling, *Beowulf Cluster Computing with Linux*. The MIT Press, 2001.

[3] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski, “Toward a Common Component Architecture for High-Performance Scientific Computing,” in *Proceedings of the 1999 Conference on High Performance Distributed Computing*, 1999.

[4] J. Rasure and S. Kubica, *The Khoros Application Development Environment*. Khoral Research Inc., Albuquerque, New Mexico, 1992.

[5] E. Seidel, “Cactus,” *IEEE Computational Science & Engineering*, 1999.

[6] K. Melero, “Open Source Software Image Map Documentation.” http://www.ossim.org, 2001.

[7] D. C. Stanzione Jr. and W. B. Ligon III, “Infrastructure for High Performance Computer Systems,” in *IPDPS 2000 Workshops, LNCS 1800* (e. a. Jose Rolim, ed.), pp. 314–323, ACM/IEEE, Springer-Verlag, May 2000.

[8] MPI Forum, “MPI: A message passing interface standard, version 1.1,” tech. rep., University of Tennessee, 1995.

[9] MPI Forum, “MPI-2: Extensions to the message passing interface,” tech. rep., University of Tennessee, 1997.

[10] A. P. Cracknell, *The Advanced Very High Resolution Radiometer*. Taylor and Francis, 1997.

[11] R. Ross and W. Ligon III, “An Overview of the Parallel Virtual Filesystem,” in *Proceedings of the 1999 Extreme Linux Workshop*, 1999 June.

[12] P. Carns, W. Ligon III, R. Ross, and R. Thakur, “PVFS: A Parallel File System For Linux Clusters,” in *Proceedings of the 4th Annual Linux Showcase and Conference,* October 2000.