Astronomical Data Analysis Software and Systems VIII
ASP Conference Series, Vol. 172, 1999
D. M. Mehringer, R. L. Plante, and D. A. Roberts, eds.

Demonstration of Parallel AIPS++ Deconvolution Application

Wes Young

National Radio Astronomy Observatory, P.O. Box O, Socorro, NM, 87801

Douglas A. Roberts

National Center for Supercomputing Applications, University of Illinois Urbana-Champaign, Urbana, IL 61801

Abstract. A parallel algorithm applicator has recently been added to the AIPS++ system. This applicator class can be used for algorithms that can carry out significant work across multiple processors with little or no communication between the processors. This demonstration will show the first use of this class to deconvolve a large spectral line cube within AIPS++. The demonstration will be running remotely on one of the NCSA Origin2000 systems. The implementation of parallel processing is with the Message Passing Interface (MPI). MPI is a portable system that allows data and instructions to be sent to remote processors (either on the same machine or on different machines). Thus, the potential speed-up demonstrated on the highest-end systems available at large centers, such as NCSA, can be realized in part on a modest multiple-processor computer systems as well.

1. Introduction

Data reduction in radio astronomy has several processing steps which are amenable to parallelization. Spectral-line deconvolution is an obvious candidate since processing can be carried out on each plane independently.

We demonstrate here a "parallelized" Clark CLEAN (Clark 1980) algorithm implemented in the AIPS++ environment.

The demonstration by necessity uses small image cubes (512×512 pixels by 100 channels) for timeliness. We make significant time-savings gains when processing large image cubes with many planes and pixels.

2. Implementing Parallelization in AIPS++ Goals

1. Use the standard AIPS++ User Interface (for familiarity).
2. Overlay a parallelization infrastructure on existing AIPS++ algorithm implementations.
3. Minimize changes to existing code.

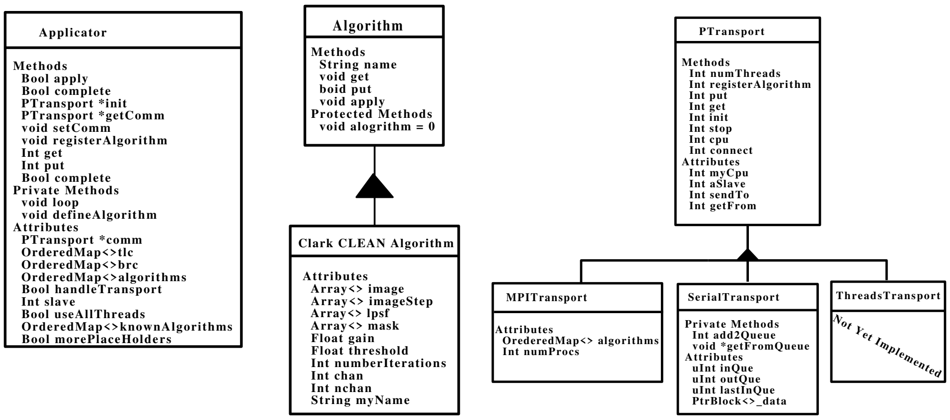

Figure 1. Object diagramas of the Applicator and Algorithm classes (*left*) and PTransport classes (*right*).

3. Design

We chose an Algorithm-Applicator scheme for the embarrassingly parallel problems. To communicate between the Algorithm and Applicators we use a PTransport object. This scheme is well suited to the AIPS++ programming environment.

The Algorithm runs n-copies of itself, each are independent of its siblings (in our case, the deconvolution of one plane in a dirty image). The Applicator is the controller, setting up the problem, sends data to the Algorithm processes, and receives the results.

PTransport is the conduit for transferring data between the Algorithms and the Applicator. Using this design provides a way to take advantage of parallel processing if it's available. Object diagrams of the Applicator, Algorithm, and PTransport classes are shown in Fig. 1.

4. Implementation

We use a simple master/slave scheme for doing the parallelization. The Applicator (pimager) does the bookkeeping and controls the IO to and from the AIPS++ system. The Algorithm (Clark CLEAN) processes receive a plane from the dirty image and deconvolution parameters from the Applicator, deconvolves the image, and send the results back to the Applicator.

The Applicator determines what type of PTtransport to use at run time, parallel, serial or in the future threads. Currently we have implemented serial and MPI PTransports[1]. A thread based PTransport will be implemented (once the AIPS++ libraries are thread-safe).

[1]While it may appear that there are n pimagers running, only one of the pimagers is a pimager object (applicator), the rest are ClarkCleanAlgorithm objects (algorithms).

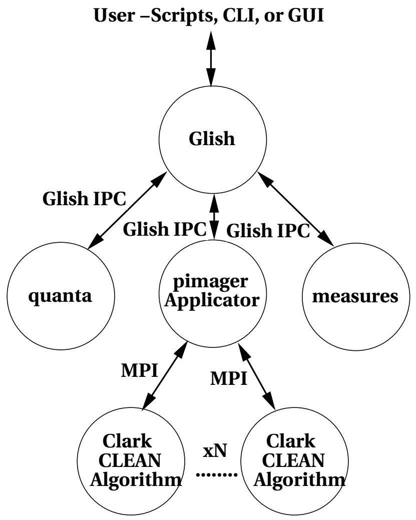

Figure 2. Applicator/Algorithm process schematic.

5. Operating Environment

Our system is evolving. To process on the "large" multiprocessor SGI systems at NCSA we create a command script and pass it into a batch queue. Notification of when the processing is complete is done via e-mail.

6. Results

Preliminary testing of parallelization in AIPS++ show promising speed ups for multi-channel deconvolution. Figure 4 shows the speed up falling away from linear at 15 processors. We are investigating why this is happening.

Dave Westpfahl (New Mexico Tech), supplied us with four pointings of VLA HI data of M33 (B, C, and D arrays) to create the images. Data in the demo are from one of these pointings.

We intend to combine the four pointings into a mosaic of 6000×6000 pixels (about two degrees on a side) by 140 channels.

Acknowledgments. Thanks to Brian Glendenning for AIPS++ and many hours of discussions. Dave Westpfahl provided the M33 datasets. Athol Kemball who provide many useful comments. We dedicate this work to the memory of our friend and colleague Dan Briggs. Blue skies Dan.

References

Clark, B. G. 1980, A&A89, 377

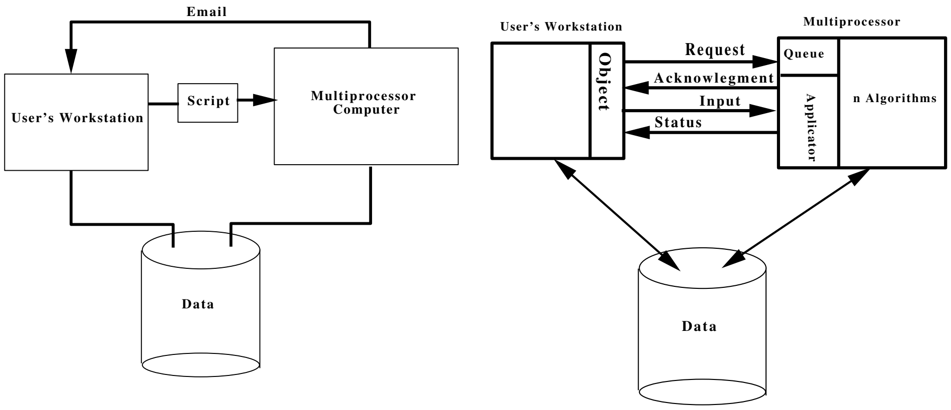

Figure 3. Caption (*left*). A more transparent scheme for interacting with batch queues (*right*).

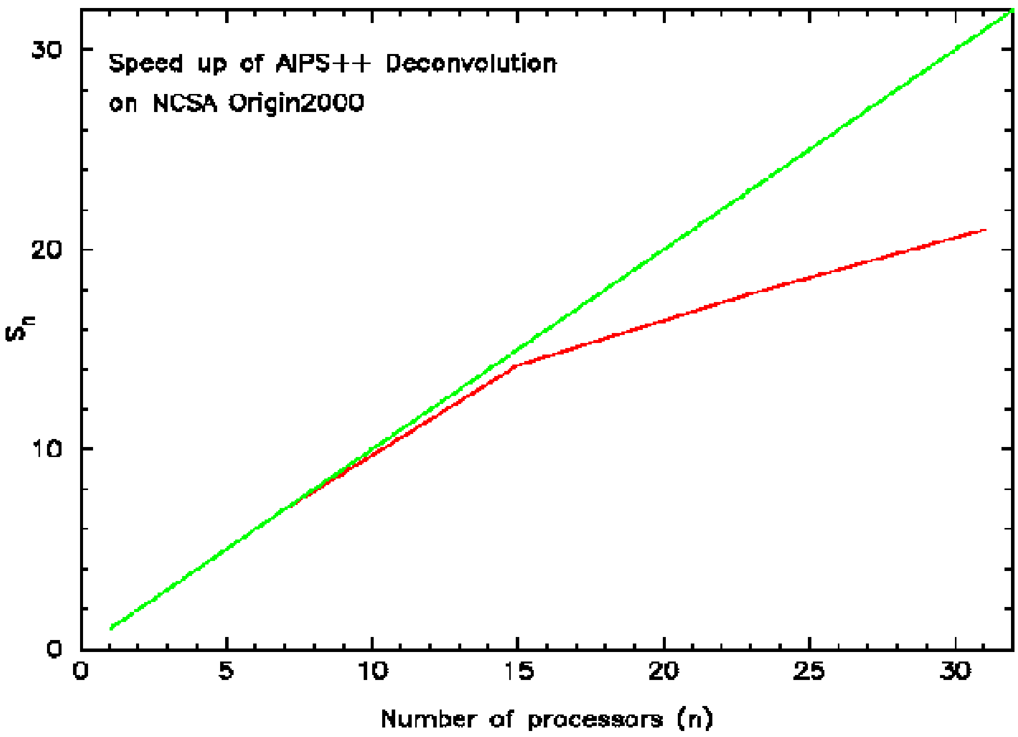

Figure 4. Clark CLEAN speed up of M33 HI 512×512 pixels by 100 channels using AIPS++ Applicator/Algorithm classes.