NOTE 203 – PARALLELIZATION OF AIPS++

Doug Roberts

October 31 1996

Contents

1 Introduction

The need to implement compute intensive astronomy algorithms on parallel machines is well known. The current AIPS++ design in C++ is not an effective environment in which to implement high performance code. However, C++ does allow calls to extrinsic subroutines or libraries that can be written in a language better suited for parallelization, such as Fortran. Additionally, tasks can be parallelized at a high level using the Glish interface. This document serves three major functions. First of all, the techniques of parallelization are discussed. Secondly, we discuss the methods appropriate for parallelization of the individual user specifications (i.e., desired capabilities of AIPS++). Finally, the user specifications are prioritized into a list of concrete recommendations for parallelization, including requirements of AIPS++ and a tentative time table.

2 Types of Parallelization

Parallelization comes in a variety of "grains." By "grain," we mean that scale of the problem that is parallelized. One extreme is illustrated by spectral-line processing. In this case, individual channels are totally independent, thus, each can be sent to a separate processor for computation. Because there is no communication between processors, communication overhead is negligible and the speed-up is nearly linear with the number of processors. This type of parallelization is easy for the programmer to implement, because the Glish interface of AIPS++ can control the spawning of processes in a parallel way. No low-level optimization is required. Problems in this category are called "embarrassingly parallel" (EP).

However, many problems are not independent at a large scale and may have complex dependencies that prevent simple Glish-level parallelism. For many such problems, it is possible to identify low-level routines, such as FFT's, that carry out the majority of the computations. In these cases, the computationally intensive parts of the code can be designed to call extrinsic Fortran subroutines or libraries. These Fortran routines can be written to take advantage of parallel architectures at a low level. This type of parallelization is called "fine-grain parallelization."

A third category is particularly difficult, in which the large-scale problem is not separable into independent processes and computational time is not spent in parallelizable low-level functions.

2.1 Embarrassingly Parallel Problems

For these problems, existing C++ code will be used whenever possible. A high level Glish wrapper will manage the execution of each code on the subsections of data. This may include some estimation of execution time and overhead for distribution of data and recombination. Ideally, this estimation should also take into account the size of the data relative to the local memory available on each processor. When estimates suggest negligible speed up, the user should be queried before execution. The long-term goal should include execution on heterogeneous machines, but this can be added after parallelization of a single machine is completed. Because the system calls to execute a process on a specific processor may be machine dependent, some of the Glish internals may need to be slightly changed for machine-specific commands. Also, the commands that are sent from Glish should turn off the fine-grain parallelization (by setting the number of threads to zero), so that fine-grain and embarrassingly parallel executions are not competing with each other. For EP problems, the large speed-up is achieved at a relatively low programmer cost.

2.2 Fine-Grain Parallel Problems

The High Performance Fortran (HPF) standard is fairly concrete now. The standard is an extension of Fortran 90 and includes a preprocessor that rewrites the input code to execute in a Single-Instruction-Multiple-Data (SIMD) parallelism. Thus, after parallelism, the same instruction is sent to all processors and the compiler divides the data across the processors. This is effectively parallelizing Fortran DO loops. Any Fortran 90 code can be compiled by an HPF compiler and conversely HPF code should compile under Fortran 90 compilers. The HPF compiler attempts to determine the independent loops and distribute data in an optimum way. HPF includes compiler directives that can force the compiler to recognize loops as independent and can force particular distribution of data across processors to mitigate slow transfers from global to local memory. Also included are pre-tuned routines for some common applications, such as FFT's. Several vendors currently have HPF compilers. The Portland Group offers compilers for a variety of platforms, from Silicon Graphics to Sun Sparcstations. Many hardware vendors are developing their own HPF compilers, such as DEC and IBM.

Although the fine grain parallelism requires a substantial initial investment of programmer time, two aspects mitigate this cost. First of all, some optimized Fortran libraries exist in the HPF libraries. Thus, parallelism is achieved with almost no additional programming. Secondly, the development of parallel libraries is a one-time project. Once the libraries exist, new programs can take advantage of the parallelism by writing them in such a use parallelized library functions whenever possible.

2.3 Combination Problems

There is a class of problems that are a combination of fine-grain and EP problems. These problems have a large amount of data dependency that prevents significant speed up with increased number of processors. If possible, these algorithms should be rewritten in a way to emphasize fine-grain or embarrassing parallelism. One purpose of this document is for programmers to keep parallel considerations in mind from the beginning, such that extensive rewriting for parallelism is unnecessary. In cases where such rewriting is not possible because of the organization of the algorithm, other solutions could be explored, such as exchange to a powerful, single-processor vector machine (e.g. Cray YMP).

3 Categorization of User Specifications into Embarrassingly Parallel, Fine-Grain, or Combination Problems

The specific user specifications have been categorized into (1) embarrassingly parallel, (2) fine-grain, and (3) combination problems. Within the EP and fine-grain cases, the problems are subdivided into other categories. Within each category, the problems are presented in the order that we propose to address them. Generally, the order is from easiest to implement to most difficult. Starting with straight forward problems gives us the opportunity to gain expertise that will be required for more complex problems. Some descriptions end in a star ($\star$); these are of particular interest because the functionality does not exist in any general data reduction package. Work on these problems should be given a higher priority in order that the parallel effort of AIPS++ coincide with the general theme of AIPS++, in which new functionality is given presidence over repetition of existing functionality.

3.1 Embarrassingly Parallel Problems

The embarrassingly parallel problems are divided into (1) general, (2) derived, and (3) CDCI cases. General cases involve problems that recur in astronomical data reduction, such as calibration. Derived cases are specific cases that are made up of one or more components of general problems. Also identified are a third category of Calculate-Distribute-Collect-Iterate (CDCI) problems. CDCI problems are a specific form of the derived case where the problem can be represented as a collection of general EP problems with a significant fraction of time spent in the collection and comparison of the distributed results in order to direct additional iterations. CDCI problems are among the most computationally intensive.

3.1.1 General Problems

General embarrassingly parallel problems will require the distribution of existing C++ code across multiple processors. Once these routines have been constructed, they can be used in other applications (see §3.1.2).

- **Spectral line image construction and deconvolution.** Spectral line processing, including imaging and deconvolution can be carried out easily in an embarrassingly parallel way. This is a common case and is easily implemented. The details of this implementation of an EP case will be important as a template for more complicated problems. This should be the first implementation.

    1. **Spectral line cube formation.** This is the simplest case, where independent spectral-line channels are sent to separate processors for imaging.
    2. **Spectral line deconvolution.** Independent spectral-line channels are sent to separate processors for deconvolution. If both imaging and deconvolution are requested by the user, the two functions should be pipelined together and sent to individual processors in one step.

- **Linear mosaic algorithm with linear deconvolution (MOSLIN in SDE) together with linear combination of pre-deconvolved images, weighting determined by primary beam.** Separate fields are independent and can be sent to different processors.

- **Antenna-based determination of calibration and self-calibration.** This problem can be separated into independent time slices and sent to individual processors.

- **Antenna and baseline-based fringe fitting for a range of spectral channels and fringe rates (normally only for VLBI data).** This is a very computationally intensive problem that can be separated in time for parallel processing. $\star$

- **Image construction from calibrated total power data (frequency-switched, beam-switched, multi-beam, focal plane array) sequences from single antennas and phased arrays, with and without spectrometers.** This can be divided into separate time ranges and sent to separate processors.

- **Calibration for non-isoplanicity using special extensions of self-calibration.** This is the general case, which includes the wide field imaging, with clusters of fields. Fields could be constructed by shifting the phases of the visibility data then sent to individual processors.

- **Parameter-driven automated flagging for large data sets.** This could be done by slicing in time. However, flagging operations are usually not computationally intensive, thus the benefit of this is not expected to be great. Low priority.

3.1.2 Derived Applications of Embarrassing Parallelism

Once embarrassingly parallel general tasks, such as calibration, are parallelized, programs to address many problems ("derived applications") can be parallelized by calling the appropriate subtasks. The order of these tasks is not as important as the general case, because these will be addressed after the general cases that they call or emulate have been written.

- **Imaging of spectral line data sets with continuum subtraction based upon continuum data or continuum models.** This is made up of continuum subtraction and imaging components, which will have been coded in the general case.

- **Self-calibration and editing of all pointings in one processing step.** This is a composite of calibration and imaging, which would have been previously parallelized.

- **3-D mosaicing allowing for sky curvature (non-coplanar baselines).** This is made up of separation of data in fields and implementing a mosaic deconvolution for each field. These operations are composites of the general operations.

- **Simultaneous, multiple field imaging with ungridded data subtraction using MX-like algorithms.** This is a straight forward task that has been previously implemented in a parallel way with PVM in SDE (Dragon). It should be straight forward to implement this code, since it has been parallelized at this granularity before.

- **For polarization calibration, all calibration sources are resolved and the polarized intensity distribution may not be like the total intensity distribution, therefore one must iteratively determine both source polarization structure and instrumental polarization.** This is a special application of calibration, which would have been parallelized previously.

- **Imaging using multiple-frequency data sets and a user-defined model for spectral combination "rules."** This separation of data into multiple frequencies is analogous to spectral line imaging, where frequencies are independent, or whose dependencies (i.e., spectral indices) are known.

- **Imaging fields larger than the isoplanatic region.** This includes the specific problem of 3-D imaging of data affected by sky curvature. The general non-isoplanatic problem is very difficult and computationally intensive.

- **VLBI imaging fields of view not radially smeared due to finite bandwidths are relatively small, so one needs "fringe-rate" imaging and multi-pointing processing for widely spaced sources in the field.**

3.1.3 Calculate-Distribute-Collect-Iterate (CDCI) Problems

Many computationally intensive problems that may be posed in a manner that in which the data are separated somehow into independent pieces and sent to individual processors. The results are brought together and the instructions for further processing are determined and the data, then new instructions (e.g. revised model) are sent again to the processors. The success of using the embarrassingly-parallel techniques on these CDCI problems is limited by the ratio of time spent on individual pieces (parallel operations) versus the I/O and calculations needed for the next iteration (serial operations). These problems are inherently computationally expensive, however, the ultimate speed up on parallel machines may vary depending on data size and algorithm. Before extensive work is devoted to parallelizing these algorithms, the estimated degree of speed up for representative data sets should be investigated.

- **Determination and correction for pointing errors and errors in beam shape, using mosaic self-calibration techniques.** These problems can be separated into short times where individual the effects of telescope pointing are determined on a baseline basis. After an initial iteration, the data are collected, imaged and a new model created for the next iteration. ⋆

- **Non-linear (MEM-based) mosaic algorithms (VTESS in AIPS, MOSAIC in SDE).** These basically involve a large number of independent deconvolutions, then a combination to create a new model, which in turn is used for the next iteration of independent deconvolutions.

3.2 Fine-Grain Parallel Problems

Problems that can be addressed by fine-grain parallelism are divided into (1) general, (2) derived, and (3) specific cases. The general cases are ones that use a large number of low-level parallelizable functions, such as FFT's. Once these problems are addressed, a library of parallel Fortran subroutines will exist that can be called in future programs. Derived problems are ones that may use a number of parallelized functions created for the general problems. Specific problems are ones that can be parallelized at a low level, such as de-dispersing pulsar data, which are very computationally intensive. However, the solutions to these problems are not generally applicable to other cases.

3.2.1 General Fine-Grain Parallelism

General fine-grain problems will require the construction of optimized Fortran subroutines. Once these routines have been written, they can be used in other "derived" applications (see §3.2.2).

- **Computation of "dirty" images and point spread functions by 2-D FFT of selected, sorted, and gridded data with user con-**

trol of data selection, gridding algorithm and its parameters, and image parameters (image size, cell sizes, polarization).

1. Optimized FFT libraries exist in HPF distribution.
2. Develop parallelized gridding and sorting libraries.

Because so many of the problems AIPS++ addresses have imaging at their core, optimization of this step will affect a large number of other user specifications.

- **Direct Fourier transform (DFT) of arbitrary (and usually small) size fields.** Optimized DFT libraries *may* exist. DFT's are not carried out often, so a great deal of attention is not warranted early on.

3.2.2 Derivative Applications of Fine-Grain Parallelism

Once general tasks, such as FFT's, gridding etc., are parallelized, programs to address many problems ("derived applications") can be parallelized by calling the appropriate parallelized Fortran subroutines.

- **Imaging after subtraction for sources.** This is basically gridding and regridding. Assuming that these libraries have been parallelized previously for imaging optimization here will not require any additional coding.

- **Image deconvolution from dirty image and point-spread function.** This includes the CLEAN algorithm (Högbom, Clark-Högbom, Cotton-Schwab, and Smooth-stabilized) and MEM (maximum entropy and maximum emptiness) deconvolution. The extensive use of FFT's (which could be parallelized) in deconvolution will offer some performance enhancement. For multiple-field or spectral-line image reconstruction, using an embarrassingly parallel construction would provide greater performance increase.

3.2.3 Special Applications of Fine-Grain Parallelism

- **De-dispersing of spectral, long time series data for pulsars with analysis and fitting in the intensity-frequency-time domain.** It may be possible to use an existing parallelized program from a pulsar group and put it into AIPS++. The time for this would be short and an additional functionality is added to AIPS++. This is a *very* computationally-intensive processes. Optimization and installation on a fast parallel computer would encourage pulsar scientists to use AIPS++ for that single functionality. $\star$

- **Briggs Non-Negative Least Squares (NNLS) algorithm.** This is one algorithm that solves the linear $A * X = B$ problem with non-linear constraints. The Briggs NNLS algorithm is one solution, however, other solutions may have been developed for non-astronomical problems. Because this class of algorithms are very compute intensive, if other groups have implemented them on parallel machines we could use them with out time-consuming code development. $\star$

3.3 Combination Problems

Combination problems should be avoided by rewriting of algorithms. However, in cases where alternate formulation of algorithms is not possible, attention should be given to estimated improvements using both fine-grain and embarrassing parallelism.

- **Cross-calibration (enforced consistency) between data taken with different instruments (flux-scale, pointing).** This includes a great deal of dependency, but computations are not likely to be spend large amounts of time in low-level parallelized routines. $\star$

- **Pointing self-calibration to determine corrections to single-dish and visibility data.** This includes much dependency and a large fraction of time spent in comparisons etc. $\star$

- **3-D self-calibration.** Spectral line channels are related to each other by the velocity structure of the observed source in the same way spatial dimensions are related. This additional information could be used in theory to self-calibrate a spectral line data set. This would allow for self-calibration of a cube where the signal-to-noise ratio in a single channel is insufficient for convergence.

4 AIPS++ Requirements and a Tentative Time Table

4.1 AIPS++ Requirements

- **Glish changes.** Changes to Glish internals to include machine-specific system calls required to execute a process on a particular processor must be carried out.

- **File access and locking.** AIPS++ must have the capability to lock files and control file access when separate processors are working on separate parts of data in the EP case.

- **Selection of f77 and HPF code.** The AIPS++ installation system must be able to select between f77 and HPF Fortran subroutines, depending on whether the site has an HPF compiler.

- **User input of number of processors.** The user interface must have the ability for the user to indicate how many processors (ultimately what machine the processors are on) to use for a task execution.

4.2 Tentative Time Table

4.2.1 Pre-1997

- Distribute Parallelization Plan to AIPS++ Programmers.

4.2.2 First Quarter 1997

- Implement calls to existing tuned HPF libraries.

- Rewrite existing Glish internals to allow machine calls to select processor numbers.

- Parallelize simple spectral line imaging and deconvolution in Glish.

4.2.3 Second Quarter 1997

- Import pulsar de-dispersion software, parallelize at fine grain.

- Parallelize sorting and gridding at fine grain.

4.2.4 Third and Forth Quarters 1997

- Parallelize antenna-based calibration and self-calibration.
- Investigate parallel solutions to linear problems with no-linear constraints (similar to Briggs NNLS).