90204

Software Environment for Parallel Processing

Sanjay Bhatnagar

****

This report discusses the various software options available for Parallel Processing at GMRT and presents a comparative study of the software environments for writing Radio Astronomy applications for parallel machines. One of the software environments have been implemented for the Param class of machines. All the software for communicating between the processors and routing data/messages within the network has been written. This will form the base on which parallel programs can now be easily developed. The task CLEAN has been parallelized and tested on a Transputer network. Roughly linear speed up has also been demonstrated.

Introduction

A significant amount of code in most image processing algorithms is to do with either user interface, the data handling, history or error mechanism. Most of the data in Radio Astronomy is stored as FITS files and therefore a significant amount of code of reasonable complexity goes into FITS I / 0 too.

All these kinds of code, though make a significant part of the entire application, have nothing to do with the actual algorithm and therefore with parallelization. Also, this code is generally common to all the applications involving image processing. In parallelizadon, most often it is the lowest level (pixel level) algorithm that needs to be recoded and ‘spawned' for the parallel processor. It is therefore. desirable to have a software decomposition in such a way that the user interface, FITS I/0 etc. and the pixel level algorithm are separate independent modules and link together with a very well defined but flexible interface.

From the project management point of view also such a decomposition with minimum dependencies between the modules, is desirable since, as long as the interface (which should be clearly defined) is strongly adhered to, different levels of software can be developed by different people, often with very different interests (a person interested in system software may not be too excited about writing a 2D PET algorithm). Working on a software where these stages are inseparably mixed, might act as mental blocks for people interested in just one of the above mentioned classes of software.

Therefore for the purpose of developing software for parallel machines at GMRT the two available software environments other than AlPS - The Software Development Environment (SDE) and Miriad/Werong were examined. I am strongly in favour of SDE being taken as the environment for development of parallel programs at GMRT (on C-DOT PPS, Param or any other machine) and have shown the workability of the SDE model by actually implementing it for the Param class of machines. Having done the ground work involving the FITS I/0, user interface, and many other support operations for images like FFTs, smoothing, etc. work can now begin directly at the parallelization level. Integration of parallel code with the rest of the SDE in a straight forward and natural manner has also been demonstrated by a full implementation of the CLEAN algorithm. It has also been shown that this software model is well suited for C-DOT machine as well and can actually deliver code which is portable across these two machines.

Why not AlPS?

The most commonly used software package in Radio Astronomy for data reduction/image processing is the Astronomical Image Processing System - AlPS. This provides, apart from the Radio Astronomy specific algorithms, almost an exhaustive list of general image processing algorithms, and image display facilities. Since this is an open software package distributed freely, many users (mainly from NRAO and Australia) have contributed to it and it has been in use for last 12 years.

To make it portable to different machines used at different sites, as a design decision, absolutely minimal assumptions were made for machine configuration and its capabilities (memory size, display and storage devices etc.). While this has actually made most of the software really portable, price has been paid in terms of software design. As more and more sophisticated computers became affordable, additions were made to the existing software, keeping the basic design same. This has resulted into an almost unmanageable and certainly unmodifiable package. The time, drive and effort required to understand **the** software structure is often many times more than the drive to contribute to the system. For a more practical point of view too, a beginner who has been trained for, and with much better, software systems, is unwilling to understand and contribute to a system which is old fashioned and is destined to be replaced by a better, more modern software. Apart from all this, it is also known that the present AlPS is on its way to elimination and will eventually be replaced by a newer design AIPS+4-.

I have myself never made any attempt to add/modify AlPS, but from the experience gained in just building the package (compiling, linking, etc.) and from the experience of other members of the group who have attempted to do so, it was clear that a better more friendly software environment has to be looked for.

Also when AlPS was designed, the concept of parallel processing for computing did not exist as it exist now. It is therefore not suited for such machines, and certainly not portable directly for the options we have at GMRT. Porting of AlPS or even apart of it will require a reasonable effort from a team in which possibly will not be too excited about working in a software environment which is old fashioned and has reached the limits of its acceptance. Also the possibility of change of machine for parallel processing over a couple of years being high (because of the fast changing face of available technology and thus of parallel processing), it almost becomes a necessity to invest time and effort in designing and planning the software effort for parallel processing, which can survive, within certain limits, a change of machine and the support software. This, I feels is impossible in AlPS.

The alternatives

Over the years, many alternative packages have been developed. From our point of view, it was more interesting to look at those which have made a mark in the area of high performance computing. Two such packages are - The Software Development Environment (5DB) and Miriad/Werong.

The SDE

SDE was designed and coded by TJ.Comwell and Pat Moore during 1986-87. Most of the system code was written by Moore during that time and most of the application level code has been written by Comwell since then. Again for the reasons of portability and acceptability by the peer group, the application level software was written entirely in FORTRAN 77. All of the system software was written in C but that constitutes a small fraction of the entire package (though the entire package strongly depends on that).

From the user point of view, SDE provides a environment, very similar to that of AlPS for data reduction. All Radio Astronomy data reduction related tasks (UVMAP, CLEAN, SELFC AL. FITs, display routines, FITS I/0, etc.) are available. The other very attractive thing about SDE is that, that this is the only package which supports 3D Imaging algorithms (which is the primary motivation behind Parallel Processing at GMRT). This makes SDE almost the ‘natural' choice for initial development at GMRT. Data is read/written as FITS files, and therefore can be imported/exported to/from AlPS or any other package which understands FITS.

Careful software decomposition has been done so as to keep the actual computation code free of code extraneous to the algorithm (this is not so in AlPS, which is one of the strongest reasons why it is almost unmodifiable). For writing code for a specialized machines, such a decomposition is extremely useful, since one can clearly see which piece of the code needs to be modified. SDE has been run on a variety of machines including CRAY.

The Miriad/Werong package

A package called Werong was written by Bobs Sault and team for data reduction (mainly spectral data) and was tailored for running on supercomputers. Application programs for this are written as stand alone programs, which are executed by a user interface, which is now called Miriad. This interface is also very similar to that of AlPS and SDE user interface.

Functionally, this also provides all the functionality of AlPS data reduction and display. For the display part, this is more similar to AlPS as it uses the AlPS display itself (SDE uses the PGPLOT library for display). The entire package is in FORTRAN (except the Miriad shell). It has extensive library for pixel/image operations, FITS I/0 etc.

Why SDE?

So as far as the user is concerned, AlPS, SDE, Miriad will be very similar.

However, from the program development point of view, Miriad is only slightly better than AlPS. It is certainly not as modular as SDE and the software decomposition is not as clear either. Each of the task is stand alone task. The handling of data files (FITS files) etc. is visible at the user level and therefore, messy.

Where as in SDE, most of the jobs which are not related to the actual algorithm are done transparently and the algorithm subroutine just links to a standard library. The task is clearly divided into three stages and the interface between these stages is simple. One can modify the code within any of the stages without effecting the rest of the application (as long as the interface is not tampered with). This gives a high level of information hiding and treats complicated files as one single objecL

SDE Implementation for Transputers

Having convinced myself for using SDE as the development environment, I planned to implement the SDB model for the Param class of machines (the C-DOT machine is not yet available for actual application experimenting). The two machines that we are using at GMRT, have different architectures. However, it would be highly undesirable to spend effort in software development and produce a software which is highly specific to one particular machine architecture (issue in software development is not to write a program and make it run somehow - it is to design it well).

Therefore, to begin with, a detailed understanding of SDE structure was soughL The documentation that is available for SDE, gives a good overview of the software, but is inadequate for understanding the software structure and its implementation. A good amount of effort went in understanding that (that was necessary to decide how to go about writing code which will easily integrate into SDE).

In a meeting on July 5 th. 1991, it was decided to go for writing code for Param class of machines for CLEAN algorithm (presumably for short term demonstration purpose). So as a diversion (which did not harm since the SOB design was more or less understood), a stand-alone task for CLEAN was written (outside SDE). A single Transputer board with one mega byte of memory, was used (which had its limitations). However code was written in such a way that it be easily integrable with SDE later. This would have been impossible without the initial understanding of the SDE software structure.

Entire code at the communication level was required to implement the philosophy of writing portable application code. This required development of code for communication between the processors and the routing of the data/messages in the network of processors. This by far took the maximum effort as the machine was new to the me and also since this part of parallel programming is most complicated and difficult to debug - synchronization is critical and deadlocks can happen easily between communicating processes. This code is now debugged and will be directly usable in any other application that will be written now. This has also made it possible to write algorithm level code which will not bind strongly to a particular machine architecture.

Minimal effort was spend in actually parallelizing the CLEAN algorithm. This was because the parallelization of the algorithm was tried long ago on sequential machine and the bare algorithm was also tested on the C-DOT machine. Therefore, the implementation of the basic algorithm was clear. The entire effort took *4-5* weeks.

Once all this was implemented (the communication/routing software and the CLEAN algorithm), attempt was made to integrate it with SDE. This required detailed digging' into SDE system level code and was another major effort. The system level code (which is all in C) had to be modified at various places. Some of the code had to be completely re-written for Transputer. But once SDE core was made to run on the Transputer, integration of the parallel CLEAN algorithm was straightforward (implementation details will be given elsewhere).

A parallel version of the Hogbom CLEAN algorithm, with its full functionality at the user level, and with the SDE user interface is now ready. This code has been tested with 4 Transputers with real data as input The center quarter of a 512x512 map was cleaned for 1000 components using *4* Transputers. The resultant clean map and the clean components were compared with that produced by the SDE CLEAN program on SUN Sparc. The results are same to the 7th. place of decimal. The difference is because of the difference in the accuracy with which floating point numbers can be represented on Transputer and SUN Sparc (truncation errors). The speed up in the execution, as one goes from one Transputer to four Transputers was roughly linear. The algorithm can be further speeded up by parallelizing a small portion of code, during CLEAN cycles, which is still sequential. This program can be now directly run on 64-256 node Param.

Conclusions

There are **various things** that **have come** out **of this exercise, some** expected, **some not so expected.**

Futility of writing **large volumes of** software **for such applications without a** design **in mind which is flexible and is** expected to last **for** some **reasonable time is one thing which is obvious. Wit** careful **coding, it is now possible to write** code **which is not too machine** specific.

In my opinion, SOB is the environment to use **for parallel processing at GMIRT. The** clear **software**

November 28, 1991

decomposition that it provides, will allow almost anyone, with reasonable prugrarnnting skills and

understanding **of** the machine to contribute to it. This **is extremely desirable.** Work can be split **between people** interested **in system software** and people interested in application software. Integration **of** any such **effort will be simple if** uniform programming **standards are followed (which will be defined elsewhere).**

The most significant **achievement of this effort has been that** this **brings** the parallel processing **project to a** stage **where most of**

the ground work **for** application **software development has been done. Work for actual** parallelization has begun and a major **effort in this** and optimization **is now possible with the involvement of** more **people. Such a** participation was earlier **difficult since clear definition of** the job **to** be done was **difficult (and this is essential to aat people and make them feel excited about it).**

In the years to come, AlPS will be replaced **by AIPS++ as the** regular data reduction package for Radio **Astronomy. If we at GMRT put in effort to write parallel code which** integrates **with 5DB, it** will be **highly** desirable to make sure that **it is also** easily integrable into AIPS4-* **later. This will require interaction with AWS-s-i. design group and with the satin of SOB. I am of the opinion that SDE should be used as the initial environment for development and a parallel effort put in Afl'S4+ to ensure that the design is suitable** for our needs.