Distributed Processing in c in Obit

W. D. Cotton January 27, 2020

***Abstract*—There are limits to the capacity of a single multicore shared memory system and for larger problems processing must be spread over nodes of a group or cluster of disjoint memory machines. This requires a different approach to dividing up the work than is optimal for vectorizatiion or over a number of cores in a shared memory system. This memo describes a conceptial design and test implementation for distributing processing over a number of processes without shared memory inside a c language program. This design uses xmlrpc for communicating control information and a globally visible distributed file system to communicate data.**

***Index Terms*—parallel processing**

I. Introduction

THE speed of individual processing elements in computers long ago began approaching physical limits (speed of light, size of molecules ...) and faster computations require some form of parallelization. These include, vectorization (SSE, AVX on modern CPUs), multiple cores with shared memory and multiple nodes with disjoint memory. Each of these require different approaches for optimal use.

Vectorization is very fine grained parallelization and requires compact organization of data in memory. Similarly, threading for performance enhancement works best with tight loops and a compact organization of data in memory. Spreading processing over disjoint memory nodes is almost by definition coarse grained parallelization and the problem needs to be split along lines that require a minimum of interaction amoung nodes.

Obit currently makes heavy use of vectorization [1],[2], [3], [4], [5], [6], [7], and multithreading [8], [9], [10], [4], [5]. The simple case of spectral line imaging using multiple nodes is described in [11]. Data from the current generation of interferometer arrays can be processed with patience and beefy workstations using vectorization and multiple threads. Instruments now in the design phase (ngVLA, MeerKAT+) will require clusters of disjoint memory systems. This memo discusses a framework for using multiple nodes inside a given c program using the Obit package [12] [1].

II. Obit Objects

Before describing how processing can be divided up amoung a set of disjoint memory nodes it is first necessary to give a toplevel desctiption of the internal working of Obit software.

National Radio Astronomy Observatory, 520 Edgemont Rd., Charlottesville, VA, 22903 USA email: bcotton@nrao.edu

[1]http://www.cv.nrao.edu/∼bcotton/Obit.html

A. ObitInfoList

Naked c is rather primitive with very limited data types and more complex ones need to be implemented. An ObitInfoList is a associative array, a set of named rectangular arrays of a given native datatype. These are similar to python dicts (but much more limited) and are widely used in Obit to pass information around. Task parameters are parsed from the input text parameter and converted into an ObitInfoList. Most high level objects (images, uv data, etc.) have an ObitInfoList member used to define and underlying disk resident data and to pass control parameters to class routines. This minimizes changes needed when new parameters are added as they are (usually) not defined in the call sequence.

B. Data Objects

There are various data classes, visibilities, images and tables that are implemented as c structs. The base data class is the ObitData from which ObitUV (uv data) and ObitImage (image) are derived. Tables are physically associated with a uv data or image. Two disk resident forms are implemented, AIPS format and a variant of (AIPSish) fits with the details (mostly) confined to the lower level interface routines. Images and uv data are passed to software via descriptions: file name and disk for FITS and, AIPS name, class, sequence, type and disk for AIPS data. These descriptions are turned into a memory resident interface object. Tables are defined by the table type and number for the associated image or uv data.

Class functions operate on memory resdent objects which are not easily passed between processes on disjoint memory platforms. There are a set of routines that convert between the memory resident structures and a description contained in an ObitInfoList. An image object can be created using ObitImage:ObitImageFromFileInfo and a uv data object via ObitUV:ObitUVFromFileInfo. An infolist describing an image or uv data can be obtained using ObitData:ObitDataGetFileInfo; this ObitInfoList can be passed across the interface to another node as described in Section III.

III. Obit implementation of XMLRPC

XMLRPC is a standard interprocess protocol for making remote calls from a “client” to a “server” using XML to pass the information between the processes. The client thread suspends until the response from the server is received. Server processes are attached to a given port number on the host cpu and a given server process is then defined by the host URL and port number. The server implements some number of named services.

In the Obit implementation, an ObitInfoList is used to pass information to and receive information from the interface with

an intermediate XML form used in the transfer. This interface is not intended to be used for bulk data which is best shared via a distributed, globally visible file system. This interface is implemented in the ObitRPC and ObitXML classes with additional features for parallel processing in ObitMultiProc. This interface is already used for Obit software to communicate with the ObitView image viewer and for remote processing.

IV. Demonstration Distributed Processing

The model of distributed processing described in this memo uses a master node with compute server processes on remote nodes. A rudimentary compute server process is in tasks/FuncContainer.c which obtains its basic information via a input text file; examples are in Obit/share/data/FuncContainerInput1.inp and FuncContainerInput2.inp. The "test" function test message logging, reads an image, computes an RMS and logs the result.

A test master node process is in Obit/share/scripts/testFuncCont.c. This has hardcoded information and can be built by copying it to the tasks directory and doing a "make testFuncCont". This process makes use of the ObitMultiProc class to manage the interface. A suitable test FITS image is in ObitView/aaa.SomeFile.fits.

The default demonstration uses 2 remote processes, on localhost but this can be changed to physically different hosts by changing the URL in testFuncCont.c. In a directory containing FuncContainerInput1.inp, FuncContainerInput2.inp and aaa.SomeFile.fits execute the commands shown in Figure IV. This starts 2 server processes watching different ports and then the master process which uses the server processes to test a simple case of passing image information to the servers and logging information back. The output should look like what is shown in Figure IV.

V. Discussion

This memo has discussed a plausible way to allow a c program to be distributed over multiple nodes of a cluster. The interface uses xmlrpc to pass commands and control information and a distributed file system for bulk data. A very simple demonstration of passing information over the interface is given.

References

[1] W. D. Cotton, "A Fast Sine/Cosine Routine," *Obit Development Memo Series*, vol. 14, pp. 1–9, 2009. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/FastSine.pdf

[2] ——, "A Fast Sine/Cosine Routine: Revenge of the Vector Processors," *Obit Development Memo Series*, vol. 37, pp. 1–9, 2013. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/FastSine2.pdf

[3] ——, "A Fast Exp(-x) Routine," *Obit Development Memo Series*, vol. 27, pp. 1–7, 2011. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/FastExp.pdf

[4] ——, "Comparison of GPU and Multithreading for Interferometric DFT Model Calculation," *Obit Development Memo Series*, vol. 35, pp. 1–5, 2014. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/GPUDFTv2.pdf

[5] ——, "Comparison of GPU, Single- and Multi-threading for Interferometric Gridding," *Obit Development Memo Series*, vol. 36, pp. 1–14, 2014. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/GPUGrid.pdf

[6] ——, "AVX2: First Look," *Obit Development Memo Series*, vol. 49, pp. 1–4, 2017. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/AVX2.pdf

[7] ——, "Notes on icc and AVX," *Obit Development Memo Series*, vol. 61, pp. 1–2, 2019. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/ICCAVX.pdf

[8] ——, "Note on the Efficacy of Multi-threading in Obit," *Obit Development Memo Series*, vol. 1, pp. 1–8, 2008. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/Thread.pdf

[9] ——, "Implementation of Threaded Image Interpolation in Obit," *Obit Development Memo Series*, vol. 5, pp. 1–6, 2008. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/Thread2.pdf

[10] ——, "Parallel facet imaging in obit," *Obit Development Memo Series*, vol. 6, pp. 1–3, 2009. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/ParallelFacets.pdf

[11] ——, "Note on parallel Processing of Spectral Lines in Obit," *Obit Development Memo Series*, vol. 2, pp. 1–6, 2008. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/Line.pdf

[12] W. D. Cotton, "Obit: A Development Environment for Astronomical Algorithms," *PASP*, vol. 120, pp. 439–448, 2008.

Fig. 1. running demo

```
$OBIT/bin/FuncContainer -port 8770 -input FuncContainerInput1.inp &
$OBIT/bin/FuncContainer -port 8771 -input FuncContainerInput2.inp &
$OBIT/bin/testFuncCont
```

Fig. 2. demo output

```
Start job 0 on thread 0
Start job 1 on thread 1
Start on port 8770
Listing Of ObitInfoList
item='outInfoFileName' type=string, dim=[16,1,1], data= 'aaaSomeFile.fits'
item='outInfoDisk' type=long, dim=[1,1,1], data=  0
item='outInfoTRC' type=long, dim=[7,1,1], data=  61  61  1  1
     1  1  1
item='outInfoBLC' type=long, dim=[7,1,1], data=  1  1  1  1  1
     1  1
item='outInfoIOBy' type=long, dim=[1,1,1], data=  1

------------------------clip-----------------------------

testFuncCont: info    20200124T092210 0124T092210 test: Started on port 8771
testFuncCont: info    20200124T092210 0124T092210 test: some logging message nwork 32742
testFuncCont: info    20200124T092210 0124T092210 test: another logging message
testFuncCont: info    20200124T092210 0124T092210 test: yet another logging message
testFuncCont: info    20200124T092210 0124T092210 test: Image RMS 0.000453
testFuncCont: info    20200124T092210 0124T092210 test: Finished
testFuncCont: info    20200124T092210 0124T092210 test: Started on port 8770
testFuncCont: info    20200124T092210 0124T092210 test: some logging message nwork 32583
testFuncCont: info    20200124T092210 0124T092210 test: another logging message
testFuncCont: info    20200124T092210 0124T092210 test: yet another logging message
testFuncCont: info    20200124T092210 0124T092210 test: Image RMS 0.000453
testFuncCont: info    20200124T092210 0124T092210 test: Finished
```