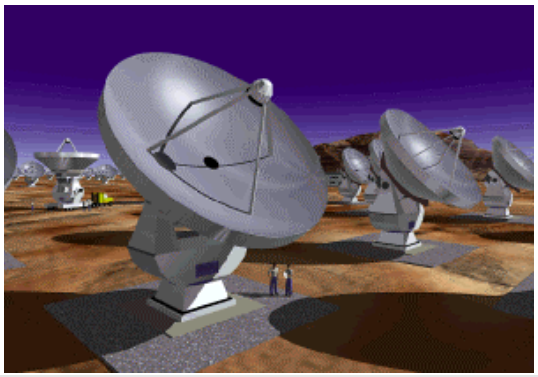

Atacama Large Millimeter Array	ALMA-SW-0NNN Revision: 1.2 2003-09-04 *Software Tests* Robert Lucas

ALMA MEMO #474: IRAM/AIPS++ Test: Phase III

Software Tests

R. Lucas, D. Broguière
IRAM
K. Golap, J. McMullin
National Radio Astronomy Observatory
R. Rusk
Dominion Radio Astrophysical Observatory

Keywords:

Author Signature: Robert Lucas	Date: 2003-09-04
Approved by: B. Glendenning, K.-Y. Morita, G. Raffi	Signature:
Institute:	Date:
Released by:	Signature:
Institute:	Date:

Change Record

REVISION	DATE	AUTHOR	SECTION/PAGE AFFECTED	REMARKS
0.1	2003-03-13	Robert Lucas	All	First draft
1.0	2003-08-30	Golap/ McMullin	All	Updated measurements.
1.1	2003-09-02	Golap/ McMullin	Section 4,5	Incorporated comments;added figures.
1.2	2003-09-04	Golap/ McMullin	All	Additional text changes from author comments.

Table of contents

1 Introduction

The main objective of Phase III of this test is to capitalize on the effort put into performing Phase I, in order to explore computing performance issues raised by millimeter-wave oriented data reduction of ALMA-sized data sets in AIPS++. It represents a first step in the monitoring of AIPS++ performance on realistic ALMA data sets that the SSR plans to define in the near future and are referred to in the off-line data processing requirements document (ALMA Software memo 18).

The data reduction functionalities that have been the subject of phase I and II are typical of present-day millimeter wave interferometry (low signal to noise calibration, high relative importance of spectral line imaging); it thus appeared worthwhile to start studying AIPS++ performance using those low noise algorithms. Also the opportunity is offered of a direct comparison of end-to-end data reduction performance (from raw data FITS files to images) with an existing operational package, as the performance of the software in current use at IRAM can be used as a starting point reference for the evaluation of further progress in AIPS++ development. The results of this test can (and already have been) used as a guide for future development, by identifying the areas where work is most needed.

Section 2 describes how the data set was built. Section 3 presents the results of processing with Gildas. Section 4 gives the results using AIPS++, and conclusions are given in Section 5.

2 Data Set Description

The data set was obtained in the following way:

1) Select an approximately two hour period of one day of the Phase I data set: (25-Mar-1997, scans 7306 to 7429).

2) Create from this data an N-antenna Plateau de Bure data set:

   a. Header data for each antenna is cloned from that of antenna N (modulo 5) of the original header data (5 was the actual number of available antennas at Plateau de Bure in 1997).

   b. All visibilities are replaced by point source visibilities, using the source flux for calibrators, and 0.07 Jy for the target source (GGTau). Random thermal noise is added corresponding to the channel width and system temperatures.

c. Antenna positions are replaced with random positions (with a dispersion of 50m, max baseline 300m), and a minimum separation of 24m. U and V are re-computed accordingly.

3) Convert the data set into ALMATI-FITS.

Four data sets were prepared with N = 8, 16, 32, 64. The data set sizes are 139, 465, 1774, and 6979 Megabytes (FITS files).

The actual integration time of the original data set was 106 min. The simulated visibility data rate is thus about 1.1 MB/s (12.5% of the specified average visibility rate, if we take into account that those FITS files contain 8-byte visibilities, not 4-byte as specified for ALMA).

3 Gildas Processing

As a validation test of the data set, and also to get a feeling of the difficulty of Phase III, we processed the data in Gildas using an automatic procedure. The data set is simple enough that the reduction is fully automatic (input fluxes are given, no phase or amplitude instrumental or atmospheric errors). The starting point is the ALMATI-FITS format, the end point is displayed images (Postscript).

The changes in *CLIC* for this test were:

1) Re-dimension it to 8, 16, 32, 64 antennas.

2) Change a few formats that would accommodate only one-digit antenna numbers.

The processing was made on a AMD 1532 MHz PC with 768 MB of memory and 256KB of cache memory (Table 1) and on bernoulli (8 GB memory, 8 PIII processors at 700 MHz, with 2 MB cache – Table 2). All data was on a local disk (60 GB, DMA enabled).

Table 1 and 2 contain the timing results (in elapsed time).

The times are split between the various steps:

1) Data filling from ALMA TI-FITS (program tifits)

2) Calibration in CLIC:

   a) Create a header file and select scan numbers

   b) RF Bandpass calibration at 3mm and 1mm

   c) Phase calibration at both frequencies, including phase transfer from 3mm to 1mm

   d) Flux calibration at 3mm and 1mm

e) Amplitude calibration at 3mm and 1mm

3) Create visibility tables using CLIC (3mm continuum, 3mm line, 1mm continuum, 1mm line).

4) Total imaging: create raw images with GRAPHIC task UV_MAP. Mapping parameters are those of phase I (256 by 256 images, 1 channel for line, 48 channels for continuum, uniform/robust weighting); clean the images with GRAPHIC task CLEAN (Clark method).

5) Total processing time.

As an indication the last three rows of Table 1 and Table 2 give:

a) The data processing rate (obtained by dividing the FITS file sizes by the total times).

b) The current SSR specified rate (doubled to take into account that the ALMA data should be recorded as 4-byte visibilities, while the FITS files used here have one 8-byte visibilities). This is scaled down by the square of the antennas for 8, 16, and 32 antenna data sets.

c) The ratio of the above.

Table 1 Gildas result on iralx7: AMD processor at 1500 MHz, 768 MB memory, 256 KB cache memory. The 'Spec' row is the SSR specified rate (however doubled as we handled 64-bit visibilities, not 32-bit as specified).

Nant	**8**	**16**	**32**	**64**
Program	**Clic08**	**Clic16**	**Clic32**	**Clic64**
Filler	18.8	86.7	214.5	815.7
Init	1.4	20.9	65.6	226.8
Phcor	3.8	24.3	108.8	405.5
Rf passband	4.9	35.8	309.7	2267.1
Phase	1.6	10.6	107.4	1301.5
Flux	3.1	7	72.2	832.3
Ampli	1.5	7.1	79.7	922.4
Total cal	16.3	105.7	743.4	5955.6
Table	6.5	39	192.9	562.3

Imaging	24.5	25.5	44.5	190.7
Total	66.1	256.9	1195.3	7524.3
FITS size (MB)	139	465	1774	6979
Rate (MB/s)	2.1	1.81	1.48	0.93
Spec (MB/s)	0.125	0.5	2	8
Ratio	16.8	3.62	0.74	0.12

Table 2 -- Gildas results on bernoulli: 8 GB memory, 8 PIII processors at 700 MHz, with 2MB cache each.

Nant	**8**	**16**	**32**	**64**
Program	**Clic08**	**Clic16**	**Clic32**	**Clic64**
Filler	30	139	595	1873
Init	4	33	201	385
Phcor	8	114	379	889
Rf passband	13	91	713	5572
Phase	3	26	328	3164
Flux	5	15	165	1900
Ampli	4	21	225	2242
Total cal	37	300	2011	14152
Table	15	67	428	1200
Imaging	31	35	68	332
Total	113	541	3102	17557
FITS size (MB)	139	465	1774	6979
Rate (MB/s)	1.23	0.86	0.57	0.40
Spec (MB/s)	0.125	0.5	2	8
Ratio	9.8	1.7	0.29	0.05

4 AIPS++ Processing

The data was processed on the same computers with AIPS++. Note: two different versions of AIPS++ were used . The tests for the first computer (iralx7) used stable version 1.8, build #499 which lacks several performance improvements in the synthesis module which are included in version 1.9, build #075 used for the tests on the second computer (bernoulli).

The tests use the almati2ms and iramcalibrater tools. The calibration stage in both Gildas and AIPS++ follows very similar routes:

1) Do a bandpass calibration.

   Gildas and AIPS++ differ slightly at this stage as Gildas (CLIC) does a phase calibration per integration time before solving for the bandpass solution; AIPS++ does a phase spline solution on the calibrater and applies that before doing a bandpass solution.

2) Apply bandpass on-the-fly and solve for phase (do phase transfer for 1 mm case before solving).

3) Establish the flux density scale.

4) Apply all of the above corrections on-the-fly and solve for amplitude.

The results for the same steps are given in Table 3 and Table 5. Note that the init step corresponds to different initialization operations in the two packages.

In Table 4, we show the ratio of processing times between the two packages on iralx7. In Table 6, we show the ratio of processing times between the two packages on bernoulli.

Table 3 -- AIPS++ results on iralx7: AMD processor at 1500 MHz, 768 MB memory, 256 KB cache memory. AIPS++ version 1.8, Build #499.

Nant	**8**	**16**	**32**	**64**
Program	**AIPS++**	**AIPS++**	**AIPS++**	**AIPS++**
Filler	114.6	346.9	1108.7	4016.1
Init	26.3	132.8	622.9	2688.4
Phcor	22.6	78.7	293.8	1280.1

Rf passband	28.5	110.3	692.9	5078.6
Phase	24	84.1	546.3	13846.3
Flux	26.8	64.8	197	871.6
Ampli	21	74.4	495.9	13558.8
Total cal	149.2	545.1	2848.8	37323.8
Table	61.2	107.4	489	2486.6
Imaging	160.7	181	384.4	1151.8
Total	485.7	1180.4	4833.1	44978.3
FITS size (MB)	139	465	1774	6979
Rate (MB/s)	0.29	0.39	0.37	0.16
Spec (MB/s)	0.125	0.5	2	8
Ratio	2.32	0.78	0.19	0.02

Table 4 -- Ratio of processing times (AIPS++/Gildas) on iralx7: AMD processor at 1500 MHz, 768 MB memory, 256 KB cache memory. Note this uses an older build of AIPS++ (1.8, #499).

Nant	**8**	**16**	**32**	**64**
Filler	6.1	4.0	5.2	4.9
Init	18.8	6.4	9.5	11.9
Phcor	5.9	3.2	2.7	3.2
Rf passband	5.8	3.1	2.2	2.2
Phase	15	7.9	5.1	10.6
Flux	8.6	9.3	2.7	1.0
Ampli	14	10.4	6.2	14.7
Total cal	9.2	5.2	3.8	6.3
Table	9.4	2.8	2.5	4.4
Imaging	6.6	7.1	8.6	6.0

Total	**7.3**	**4.6**	**4.0**	**6.0**
Rate ratio	0.14	0.22	0.25	0.17

Table 5 -- AIPS++ results on bernoulli: 8 GB memory, 8 PIII processors at 700 MHz, with 2MB cache each. AIPS++ version 1.9, Build #075.

Nant	**8**	**16**	**32**	**64**
Program	**AIPS++**	**AIPS++**	**AIPS++**	**AIPS++**
Filler	346	1777	2760	10939
Init	48	215	649	2140
Phcor	58	226	864	3484
Rf passband	36	143	519	2298
Phase	21	77	260	1111
Flux	52	147	475	2093
Ampli	16	50	142	614
Total cal	231	858	2909	11740
Table	84	257	991	5150
Imaging	128	183	243	750
Total	789	3075	6903	28579
FITS size (MB)	139	465	1774	6979
Rate (MB/s)	0.18	0.15	0.25	0.24
Spec (MB/s)	0.125	0.5	2	8
Ratio	1.44	0.3	0.13	0.03

Table 6 -- Ratio of processing times (AIPS++/Gildas) on bernoulli: 8 GB memory, 8 PIII processors at 700 MHz, with 2 MB cache each. AIPS++ version 1.9, Build #075.

Nant	**8**	**16**	**32**	**64**
Filler	11.5	12.8	4.64	5.84
Init	12.0	6.52	3.23	5.56
Phcor	7.25	1.98	2.28	3.92
Rf passband	2.77	1.57	0.73	0.41
Phase	7.0	2.96	0.79	0.35
Flux	10.4	9.8	2.88	1.10
Ampli	4.0	2.38	0.63	0.27
Total cal	6.24	2.86	1.45	0.83
Table	5.6	3.84	2.32	4.29
Imaging	4.13	5.22	3.57	2.26
Total	**6.98**	**5.68**	**2.23**	**1.63**
Rate ratio	0.15	0.17	0.44	0.60

5 Conclusions

The principal conclusions follow. Further benchmarking and performance evaluation is continuing through a targeted automated benchmarking program which provides the time evolution of data handling in several data reduction packages with regression and quality-metrics applied. See: http://shiraz.drao.nrc.ca:8080/AlmaDRPBenchmarks/.

1) The total processing rates with present-day desktop equipment are not far from what will be needed when ALMA gets 8-16 antennas. We will have to count on hardware progress (Moore's law factor is about 100 from 2002 to 2012) to process 64 antenna data (this is not a surprise, and the plan is also to have Linux clusters for the pipelines). Nevertheless we all know that the specified data rate is very likely to increase.

2) The key measurement is that AIPS++ is slower than Gildas by a factor of 1.6 for 64 antennas on Bernoulli and a factor of 6 on iralx7. The different values are

caused by the memory demands during calibration which causes significant swapping if there is inadequate memory. AIPS++ currently requires more memory (peak usage for 64 antennas is ~1 GB for Gildas and ~1.7 GB for AIPS++); this is due to a bug in the AIPS++ bandpass calibration.

This factor increases for smaller array sizes however; a number of points should be emphasized:

a) The Gildas measurements for different array sizes were based on recompiling the Gildas software for the specific maximum number of antennas (8, 16, 32, or 64). If the same 64-antenna version is used, the Gildas timings increase for smaller array sizes. The main cause for this increase is expected to be an inefficient use of cache memory as most work arrays are not dynamically allocated in the current version of CLIC, and are thus highly oversized when the 64-antenna version is used with a small number of antennas.

The AIPS++ measurements for all numbers of antennas are based on a single compiled version. The improvement in performance of AIPS++ for higher numbers of antennas is a feature of the tiling I/O access which becomes advantageous for larger data sets.

b) We have included the timings from the iralx7 machine which are accurate for the March build of AIPS++. Significant performance improvements have been made since that date. As mentioned, the bug in AIPS++ calibration will cause smaller memory machines to have worse performance for AIPS++ than measured on bernoulli at the date of this report.

c) The Gildas versions tested on bernoulli were obtained by copying binaries built on another machine. Some performance improvements have been achieved by optimizing a local compilation.

3) The basic core algorithms for the most CPU heavy operations (rf passband, phase, amplitude calibrations) are the same for both packages (essentially two Fortran subroutines, *polyant* and *splinant,* that do antenna based polynomial and spline interpolation for amplitudes and phases). The overall differences in performance are dominated by the filler and init steps which are not comparable between the two packages, handling data organization rather than numerical calculations. On bernoulli the data filler uses more than 35% of the total processing time. The ALMATI-FITS format is an interim data structure so no optimization will occur on these fillers. The data processing needs (numerical calculations) are dominated by the calibration for both packages.

The *table* step also shows a large difference in performance between the two packages. In the both packages, this step involves the application of the calibration to the data; in the AIPS++ package, this step also involves writing a subset of the data out to UVFITS format and reading this data back in as an MS (with optional averaging and a re-splitting of the data following for the continuum case); these manipulations are necessary because of the absence of a ‘split’ function to enable simple creation of a subset MS within AIPS++ (in current development plan).

4) *phcor* and *flux* are, in AIPS++, implemented using Glish. It is thus expected that they should be much less efficient than similar ones implemented in C++.

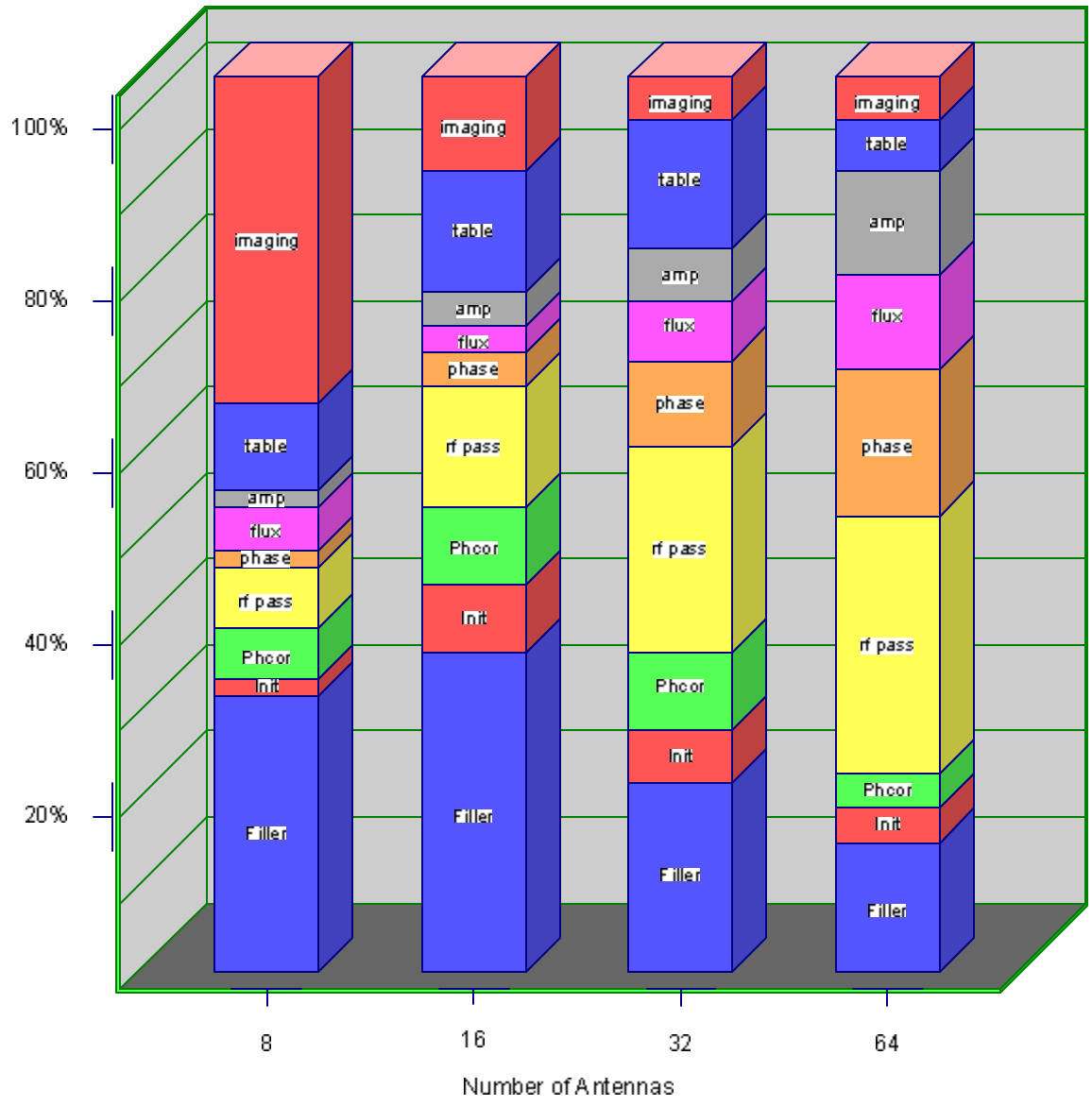

Figure 5-1 -- Gildas relative processing times on bernoulli.

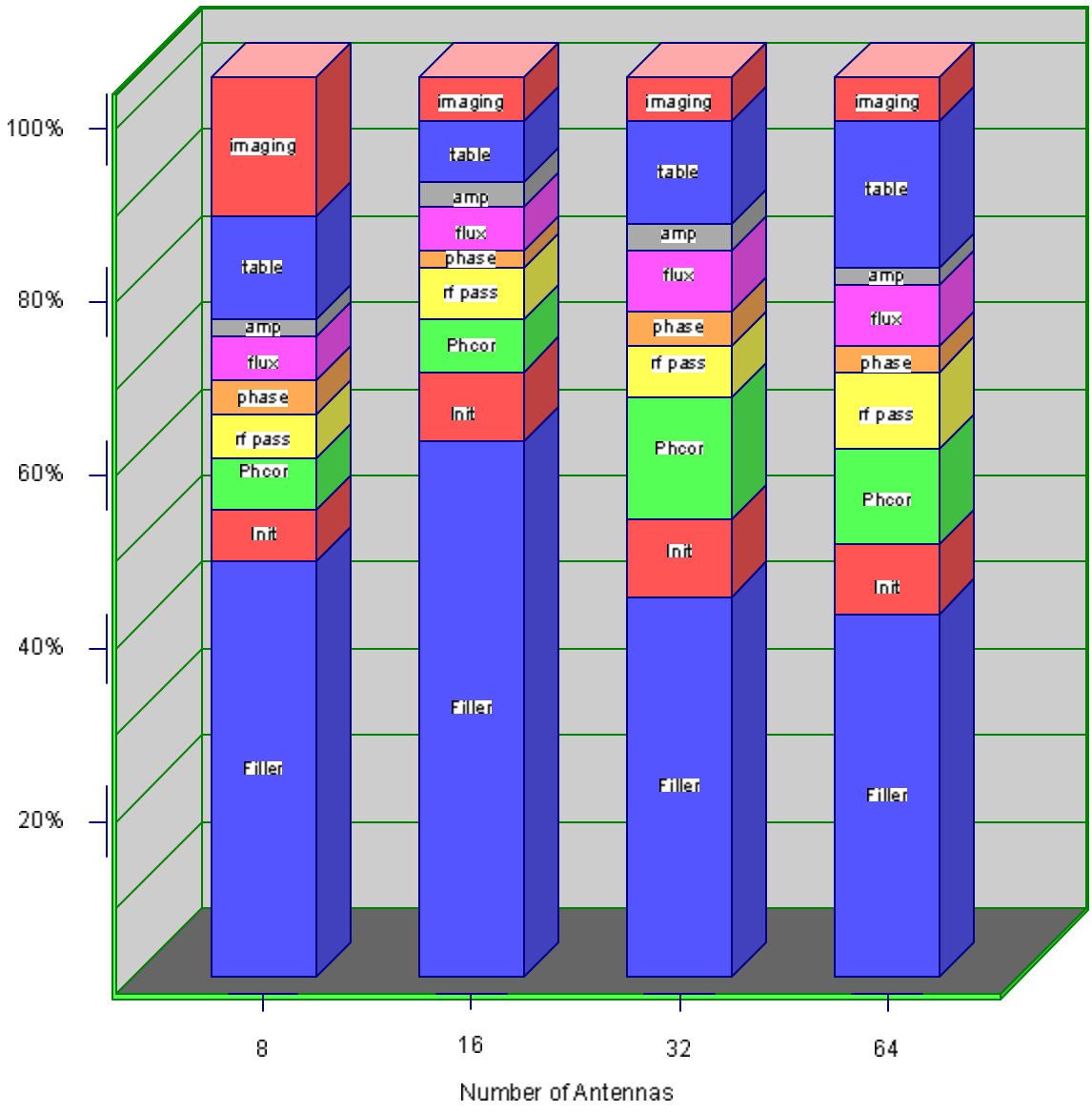

Figure 5-2 -- AIPS++ relative processing times on bernoulli.