Use of Shared Memory in Automated Pipelines

F. Masci, 3/24/03

I. Objectives

1. Main objective: minimize/eliminate use of physical disk I/O when passing intermediate FITS files in a BCD pipeline by using static shared memory buffers. The aim is to make the BCD processing as "CPU-bound" as possible, justifying the need for faster CPUs.
2. Convert existing science super-wrappers to use pre-defined shared memory buffers via a script and a configuration file which maps all FITS filenames to *cfitsio* shared memory segment ID's (see below).
3. Implement a robust design which allows multiple instances of pipelines (jobbers) to be executed simultaneously on any given drone.
4. At the end of a pipeline, have a script (library routine) which copies the contents of selected shared memory buffers (desired intermediate products) to the sandbox and flushes all memory buffers.

II. Conversion of Pipeline Super-wrappers

Assumptions:

1. It is assumed that exclusive use of shared memory ***will not*** remove the need to run multiple jobbers on individual drones. The processing time is assumed to not scale in exact (inverse) proportion to CPU speed. Thus, the design below allows for multiple pipeline instances per drone to maximize throughput. We can of course go with one jobber per drone if the new design shows little dependence of throughput on number of jobbers.
2. There will never be more than 3-jobbers running at any given time on a drone. The design below is flexible enough to allow more/less jobbers if desired.

FITS-to-Shared Mem. Map File:

Each science super-wrapper specific to an instrument/channel will have an associated configuration file or translation table which maps intermediate product FITS files to pre-defined shared memory segments. The following is a prototype "ShmemMap.tbl" file.

```
\character comment = FITS file to Shared memory translation table
\character comment = read by FITS2SharedTrans.pl
\character instrument = MIPS
\int       channel  = 1
\character basesuperwrapper = w_mipe24_sur_science_superwrapper.pl
\character sbx_copy = 1 if want to copy to sandbox, = 0 if not
\character shmem_jobID1 = shared segments for jobber 1 (range 0-31 )
\character shmem_jobID2 = shared segments for jobber 2 (range 32-63)
\character shmem_jobID3 = shared segments for jobber 3 (range 64-95)
|FITS_file      |shmem_jobID1 |shmem_jobID2 |shmem_jobID3 |sbx_copy
|char           |int          |int          |int          |boolean
 tranhead.fits   shmem://h0    shmem://h32   shmem://h64    1
```

```
cvti2r4.fits    shmem://h1    shmem://h33   shmem://h65    0
desatslpe.fits  shmem://h2    shmem://h34   shmem://h66    0
linearize.fits  shmem://h3    shmem://h35   shmem://h67    0
dmask.fits      shmem://h4    shmem://h36   shmem://h68    1
dntoflux.fits   shmem://h5    shmem://h37   shmem://h69    1
  .                .             .             .           .
  .                .             .             .           .
  .                .             .             .           .
```

<u>Conversion Script:</u>

There will be a script which replaces FITS filenames to equivalent shared memory segment ID's (as defined in the table above) in a science super-wrapper. The script will have the following prototype.

```
FITS2SharedTrans.pl  –w <super_wrapper.pl> -t <ShmemMap.tbl> -m <f or r>
                     -p <jobber ID>
```

where the inputs/outputs are as follows:

–w <super_wrapper.pl>:	Input science super wrapper or pipeline script (required).
–t <ShmemMap.tbl>:	Input FITS-to-shared map file (see above; required).
–m <f or r>:	Flag to perform forward "f" (FITS-to-shared) or reverse "r" (shared-to-FITS) translation (default = f).
–p <jobber ID>:	The jobber ID number (1, 2 or 3). This may be obtained from the PMID environment variable value which is a concatenation of the pipeline machine number and jobber ID (required).

III. <u>Method 1:</u> Dynamic conversion according to Jobber ID

If one desires to run multiple jobbers per drone, it must be ensured that the pipeline instantiated by a given jobber (with ID encoded in the PMID) will only access it's own pre-allocated shared segments as defined in the translation table above. The *first method* performs the FITS-to-shared segment replacement in the super-wrapper script dynamically prior to executing every instance of "run_pipeline.pl". This is the simplest design, although it will add some (negligible) overhead.

IV. <u>Method 2:</u> Static multiple Super-Wrappers according to Jobber ID

The second option assumes that separate jobber-dependent (FITS-to-shared mapped) science super-wrapper scripts have been created beforehand on disk and named according to the jobber ID. For example: mips24_super_wrapper1.pl, mips24_super_wrapper2.pl and mips24_super_wrapper3.pl (assuming a maximum of three jobbers per drone will ever exist). Each of these scripts execute the same science thread except that each will contain its own shared segment range as defined by the translation table above. This design will only require run_pipeline.pl to be modified to execute the correct "mips24_super_wrapper#.pl" script according to the jobber ID # (possibly through the PMID value).

V. Copying to Sandbox and Flushing Shared Memory Segments

We envisage there will be script (or perl library routine) called at the end of each pipeline to:

1. Convert desired shared memory segments into corresponding FITS files and create them directly on the sandbox disk, and,
2. Flush all pre-existing shared memory buffers.

At the time of writing, it is unclear whether the above functions should be implemented in C or perl. If implemented in C (likely more efficient), the module will take a minimum of three arguments: <SandBoxPath>, <ShmemMap.tbl> and <jobber ID> which respectively represent the full sandbox path, FITS-to-shared map file (see above) and the jobber number respectively. The module will make use of the "sbx_copy" column in the ShmemMap.tbl file to select those FITS files to physically create on the sandbox.