

**MODIS SEMIANNUAL REPORT
- JUNE 1992 -**

**UNIVERSITY OF MIAMI
RSMAS/MPO**

DR. ROBERT H. EVANS

NAS5-31362

=====
Due to the interlocking nature of a number of projects, this and subsequent reports will contain coding to reflect the funding source. Modis funded activities are designated with an M, SeaWIFS with an S, Pathfinder with a P, and Headquarters with an H. There are several major sections within this report; Database, client/server, matchup database, and DSP support.

- A. NEAR TERM OBJECTIVES**
- B. OVERVIEW OF CURRENT PROGRESS**
- C. FUTURE ACTIVITIES**
- D. PROBLEMS**

A. NEAR TERM OBJECTIVES

A.1 Modis Objectives (M)

1. Continue to develop and expand the processing environment
2. Begin extensive testing using global CZCS and AVHRR GAC data with database processing to test the following:
 - a. algorithm capability
 - b. machine and operating system stability
 - c. the functionality required for the processing and analysis environment

A.2 SeaWIFS Objectives (S)

1. Continue testing of processing methodology.
2. Continue to develop relationship between database and *in-situ* environment.

A.3 Pathfinder Objectives (P)

1. Expand matchup database as applicable.

2. Continue testing of methodology.

A.4 DSP Objectives (H)

1. Continue testing of processing methodology.
2. Continue to expand sites supported.
3. Expand the supported hardware/software platforms

B. OVERVIEW OF CURRENT PROGRESS

B.1 Automatic Processing Database (S)

B.1.1 Early 1992 Status

During the previous six months, substantial changes have occurred in the database processing. Both the FORTRAN interface to the automatic processing database and the schema for the database itself (here named AUTOPROC) have undergone extensive changes, including, but not limited to, streamlining and enhancing the database, eliminating inefficiencies in the interface and adding functionality.

In January, 1992, the automatic processing had been adapted for use with Version 5 VMS, and the programs and subroutines were working similarly to the CZCS processing project several years ago. The record-addition process had been modified, but the basic methodology and execution of the FORTRAN interface had undergone only minor changes. Many of the non-source elements (file used for database creation and definition, etc.) had changed in format, but not substantially in content. During January, the system was tested using AVHRR data. MAIN and PROCESS_CONTROL records were added using an adaptation of the TIROS ingest program, and these records were automatically processed, using 2CHAN for atmospheric correction, and REMAP for the space bin program. The source files were consolidated and the database creation and definition files were organized. In the first part of February, work was begun to generalize database methods to an IN_SITU database, but only preliminary work was completed. This work is explained further in Section B.3.

In late February, Teresa Larsen (SeaWIFS, GSFC) visited RSMAS. During her visit, a plan was developed to reorganize the database schema. Extensive use of look-up tables in the old schema had been intended to conserve disk space at the expense of processing

efficiency. Realizing that this would require extensive changes in the FORTRAN interface, we nevertheless decided that the changes were warranted. We also decided to convert the database definition files from DEC's proprietary database language, RDO into the more flexible SQL, allowing the files to be used in non-DEC database management systems (RDBMS's).

As an example of this change, the MAIN table (used to track satellite scenes) previously contained a field called SENSOR_CODE that contained a 2-byte integer which pointed to the correct satellite sensor in a SENSOR table. When sensor information was needed, the SENSOR_CODE would be extracted from the MAIN record, then a separate query performed to retrieve the character name from the SENSOR table. The previous schema (for both satellite and in-situ data) made use of approximately 40 look up tables.

Nearly all of these codes have been eliminated from the present schema. When a code was 'merged' into a table, the FORTRAN interface that referenced that code also needed to be changed. All references to the code had to be replaced with the character variable. This merging was performed during March and April. The database tables needed for the satellite processing contained approximately 17 of these codes, which were individually changed and tested. To insure that changes were made correctly, all software was recompiled, relinked and a test was run on the automatic processing. By the end of April, the FORTRAN interface had regained its former functionality, albeit with the new schema.

In May, several substantial changes and enhancements were made to the interface, including the addition of a client/server capability. The client/server is covered in Section II, and will be assumed to be a transparent information pipeline between programs and the FORTRAN interface. All changes discussed in this section are within the database interface itself, although some were made to facilitate the use of the client/server.

The most significant change was the breakup of the db_control subroutine, into two subroutines. This resulted in routines that performed the tasks more efficiently. The subroutine had been called repeatedly during the progress of a batch automatic processing job that magnifies the increased efficiency. The major functions of the old db_control subroutine were to:

- 1) when requested, supply the next processing step for an automatic processing batch job
- 2) report back the end status of automatic processing jobs
- 3) keeping the database updated on the progress of the first two.

First, the major functions were split into two subroutines: `db_request` supplying the processing steps and parameters at the beginning of an autoproducting job, and `db_report` notifying and updating the database at the end of a job. (These two subroutines serve as a touch-point between the client/server and the interface; that is to say, the client/server contacts the interface through calls to these subroutines, and another to add records under current development.)

The `db_request` function has been changed substantially. Previously, a batch job repeatedly called `db_control`, obtaining only one `process_step` and the associated workspace variables at a time. Each time, the calling program had to invoke (i.e., attach to) the database, and `db_control` updated the database after each `process_step`. In addition, all procedure names and workspace variables had been passed back to the calling program, which would have made the client/server connection quite complicated. `Db_request` avoids all of these problems by retrieving all process steps and associated parameters, and using them to construct a DSP command file that is applied to the input file. Very few parameters are passed back to the caller. The use of an NFS-mounted disk (that is to say, a disk that can share ASCII files between UNIX and VMS computers) permits the database interface to reside on VMS, while allowing UNIX processing of the data.

The second major function, performed by `db_report`, occurs at the end of a batch job to notify the database of the final status of the job. When `db_request` creates its DSP procedure file, the last line calls a program to return status. First, `db_report` evaluates the status variable and error message (if any) to determine the success or failure of the job, then updates the database accordingly. As you will see later, some very specialized functionality has been added to this subroutine.

Both `db_request` and `db_report` are fully functional in the client/server; using a VMS Server (currently required by the FORTRAN interface), either a UNIX client or a VMS client can perform requests and make reports. We are currently working on a similar

client/server function that adds MAIN and PROCESS_CONTROL records to a database. The 'touch-point' here is a subroutine called add_main_pcr (to add main and process control record). It is currently called by a version of the CZCS ingester, which extracts needed data from the raw input file, formats the information, and calls add_main_pcr to store the data in the database. The process will be extended to programs that extract the data from other types of raw files (such as TIROS or SCRIPP format tapes) as well as DSP image files.

B.1.2 Progress on the Automatic Processing Database

Specialized fields have been added to the MAIN and PROCESS_CONTROL tables to allow jobs to affect and even trigger other jobs. The first two functions that have been added are the parent/child records, and the daily mosaic process.

Each input source file has one associated MAIN record in the database. However, the data file may be too large for convenient processing. The ingesters have the capability to ingest only part of a file, e.g., a range of scan lines. The database schema and record addition process have been adapted to account for this. We term this a parent/child relationship, where one input source file (the parent) may have multiple ingested files (the children). When notified that a parent/child condition exists (as explained later), the ingester adds a MAIN record for the parent source file (if none exists), adds one PROCESS_CONTROL record (PCR) corresponding to a job to 'put the pieces back together,' marking this PCR 'HOLD,' and then creates one PROCESS_CONTROL record for each piece of a pass to be ingested. The database tracks the range of scan lines for each piece, and how many 'child' records have been created for each input parent file. As each PCR is processed, part of the reporting procedure (db_report, mentioned earlier) is to increment the 'process_done' field in the MAIN relation, and to release the 'put it back together' job when all child records have finished.

There are also situations where multiple source files may be required for a particular reason; global daily mosaics are examples of this. There will be many source files that must be processed before the full mosaic is created. In this case, there will be one MAIN and process_control record for each source file submitted for processing, and there will be one MAIN and PCR, marked 'HOLD' for the mosaic task. RELEASE_LINKS tie the jobs together that process

the individual passes and the global mosaicing. As with the parent/child case, these links are checked and updated as part of the db_report process, and when all individual source files have completed processing, the mosaic job is automatically released and submitted for processing.

In May, major improvements were made the method of assigning job priorities. Each process_control record is assigned a processing priority, and when the database is asked for the next job (the db_request function, discussed earlier), process control records with a lower priority are selected first. (Hence, a priority of 1 is the most urgent.) This functionality has been expanded and made more flexible. In addition to selecting the next job on the priority, an additional sorting key, record number, was added. Thus, among jobs of equal priority, the earliest record would be selected for processing. Further, the method of assigning the priority has greatly changed. Previously, a processing priority had been hard-coded into the ingester used to create the MAIN and process_control records. Currently as a process_control record is added, it is assigned a PROCEDURE (which represents a set of processing steps). Thus, a default priority is stored in the database for each procedure that has been defined, and this default value will be used unless overridden. If a different priority is desired for a particular file or set of files, it is easily defined in the ingest setup process used to define other database information.

For example, consider three procedures, one that is to be used on AVHRR GAC data, one to process CZCS data, and one to create a daily mosaic of CZCS data. The database procedure definitions would be:

PROCEDURE_NAME	PROCESS_MODE	PRIORITY
GAC_PROCESS	BATCH	3000
CZCS_PROCESS	BATCH	2500
CZCS_MOSAIC	BATCH	2000

In this case, CZCS processing is considered 'more important' than GAC processing, and the mosaic process takes precedence over the individual files. That is, once one day's worth of CZCS is processed, the mosaic would run next to allow the individual files to be removed from the disk. If, however, you needed a particular GAC pass processed immediately, a simple logical assignment of the

desired priority prior to record addition would be used to insure the 'rush' job would be the next queued up for processing.

One additional enhancement to how the job priority is handled occurs during the job-reporting process. In those cases where jobs are held until other jobs finish (the parent/child or mosaic jobs), the number of jobs remaining before the release is checked. If there are two or fewer, the priority of the remaining jobs is adjusted so they will be selected before others of their class. This allows the 'put together' or mosaic job (as the case may be) to run and the permits earlier removal of the constituent files.

B.2 Client/Server Status (S)

B.2.1 Early Client/Server Model

B.2.1.1 Client/Server Concept

The most commonly used paradigm in constructing distributed applications is the client/server model. In this scheme, client applications request services from a server process. Nominally, a server provides network services; a network service is a collection of one or more remote programs. These remote programs implement one or more remote procedures. The client/server operation is based on a known set of conventions that must be implemented at both ends of a connection before service may be rendered (and/or accepted). This set of conventions comprises a protocol that may be symmetric or asymmetric.

RPC, Remote Procedure Call, is used to implement the client/server model. RPC is a high-level communications program that allows network applications to be developed using specialized procedure calls, providing a degree of independence from the underlying networking mechanisms. The RPC model is similar to a local procedure call model in that one thread of control logically winds through two processes -- the client's process (the caller) and the server's process (the procedure called). The reliability of an RPC model depends on the reliability of the transport protocol underneath it. For this reason, this implementation RPC is running on top of TCP/IP.

The caller sends a message containing the required parameters for the requested procedure to the server process and awaits the results. On the server side, a process is dormant waiting the arrival of a call message; upon arrival, the server process activates, extracts the procedures' parameters, computes the results, sends a reply message, and is deactivated. Thus the server process activates, services the client request, and performs whatever appropriate actions the client requested. Once the reply message is received, the caller's execution resumes and the results are processed.

B.2.1.2 Early Client/Server Model

The early mcp model consisted of a database and its attendant FORTRAN programs, a server (on VMS), an mcp program, a dbbat, and a client (on UNIX). It was designed to work as follows:

When mcp starts, it enters an endless loop. In this loop, it will periodically call resource monitor to determine the current system resource status. If there are enough resources available, a new dbbat process is started to process a new data file. If resources are not available, the mcp process will suspend until its new activation cycle.

(The monitor is part of the future implementation; a dummy routine is in place that will always return sufficient resources.)

The dbbat process will request, using the client/server mechanism, the next available task, if any, in the database. The request is sent to the VAX through a client process. The parameters coded in the request message include server's name and service type.

Service type specified the service expected from the server. The two main services were retrieving the job steps from the database and updating the database after job termination. When a server is activated, it provides the requested service to the client by running the procedure that contacts the database.

The client/server was tested on the RDO defined database as well as the newly SQL defined database. When a client requests a task, the server will invoke the database and then fetch all the steps that are to be done and pass them to the client in the reply message. The client then will decode the message and write the job steps into a .dsp file. This .dsp file will serve as a command procedure for dbbat

process. Dbbat requests a job through client and gets a command procedure to be executed.

Our early tests used a CZCS data file and three command procedures, processanly.dsp, processnewbin.dsp and processmosaic.dsp. Some appropriate steps reflecting these three command procedures had been loaded into database. dbbat requested and received these job steps from database through client/server. These steps were then manually executed.

B.2.2 Current State

Our mcp client/server model continues to be a distributed application over a VAX/VMS and UNIX network. The database is currently VAX resident; the remainder of the mcp system resides on UNIX. The connection between these two parts is accomplished with client/server mechanism. Although there has been some redesign since the initial implementation, the basic concept of the hybrid model for the interim system has not changed. The most significant changes in the mcp structure have been made for simplicity and reliability. dbbat has been eliminated; mcp will perform dbbat's function in a more direct manner. From a functional perspective, the main changes are additions to the automated processing; these additions include the addition of records to the database and ingesting.

The current mcp model has the following parts:

- a. database.
- b. database supporting FORTRAN programs.
- c. server on VMS.
- d. MCP (includes client(s)) on UNIX.
- e. client(s) on VMS.

The system will function with a single server and multiple clients where clients can be either UNIX or VMS resident. Some care has been exercised to ensure that while executing multiple mcps with multiple clients there is no duplication of file names.

When mcp starts, it will enter an endless loop. In each loop, MCP will carry out the following:

1. It first will create a client to request a job from a specified database on a specified server.
2. If no jobs are pending, MCP will suspend for a period, the client is deleted, and the process begins when mcp is activated the next cycle.
3. mcp will invoke the shell script whose name is passed back from the server to the client (and to mcp).
4. The shell script invokes the DSP command procedure from inbound NFS disk.
5. At the completion of the steps contained in the DSP command procedures, the last step calls db_report that creates a file containing information for updating the VAX database. Control passes to the shell script that returns a status to MCP.
6. MCP reads the information file created by db_report. This file contains status and information about the execution of the DSP command file; i.e, database name, server name, record, job status, steps, completion, etc. Using this information MCP connects to the VAX server to update the database.
7. MCP returns to the beginning of the loop.

During execution, the server will wait for a request from a client. When a request is received, the server unpacks the data structure passed over from client and uses the information to call dbconnect. The result and status returned from dbconnect are checked and returned to the calling client. Depending on SVCTYPE [SVCTYPE is a parameter defining the type of service the client is requesting from the server], different programs are called from the FORTRAN supporting programs. dbconnect calls db_request to get new job steps, db_report to update the database, and db_add to add new records to the database.

Some of the significant modifications that have been made to the client/server implementation are:

1. The client has been separated from dsp. It was unnecessary to maintain this relationship.

2. The server fetches all steps required to process the input in the form of a single .dsp file to shared memory.

3. Two steps have been added to the automatic processing; add records and ingesting. The ability to add records has been added by the creation of an additional client to manage an automated process from the jukebox to the auto process. Ingest is a separate process consisting of several programs. The entire process begins by fetching file from the jukebox and ends with the final mosaic image.

4. Error handling is being enhanced; a dummy error_handler has been added that merely echoes the message received and may exit if necessary. At this stage there is no corrective action taken; a more complete error handler can be added to take these corrective actions in the future. The error handler is modular in nature so that its extension to additional functions is clear and concise.

5. Both batch and job queues were tried and rejected due to the complications in message passing. In the current implementation, the simplicity of direct communication was selected. When a task completes, a status is returned directly to mcp. Based upon the results of the status message, mcp will either perform the next step (successful completion) or evoke an error handler.

6. As part of the testing process, three and ten day CZCS data runs have been made.

B.3 Matchup Database (P)

The purpose of this section is not to restate the details of the construction of the satellite and *in situ* sea surface temperature (SST) matchup database. Such details have been provided in the previous reports. Instead, this write-up will concentrate on a critical evaluation of the progress made in the last six months, and of the lessons learned during this period. Briefly, the objective of this part of the project is to build a co-temporal database of satellite and *in situ* measurements, to be used in the development and validation of

SST algorithms. The main steps involved in this process are shown in Figure 1.

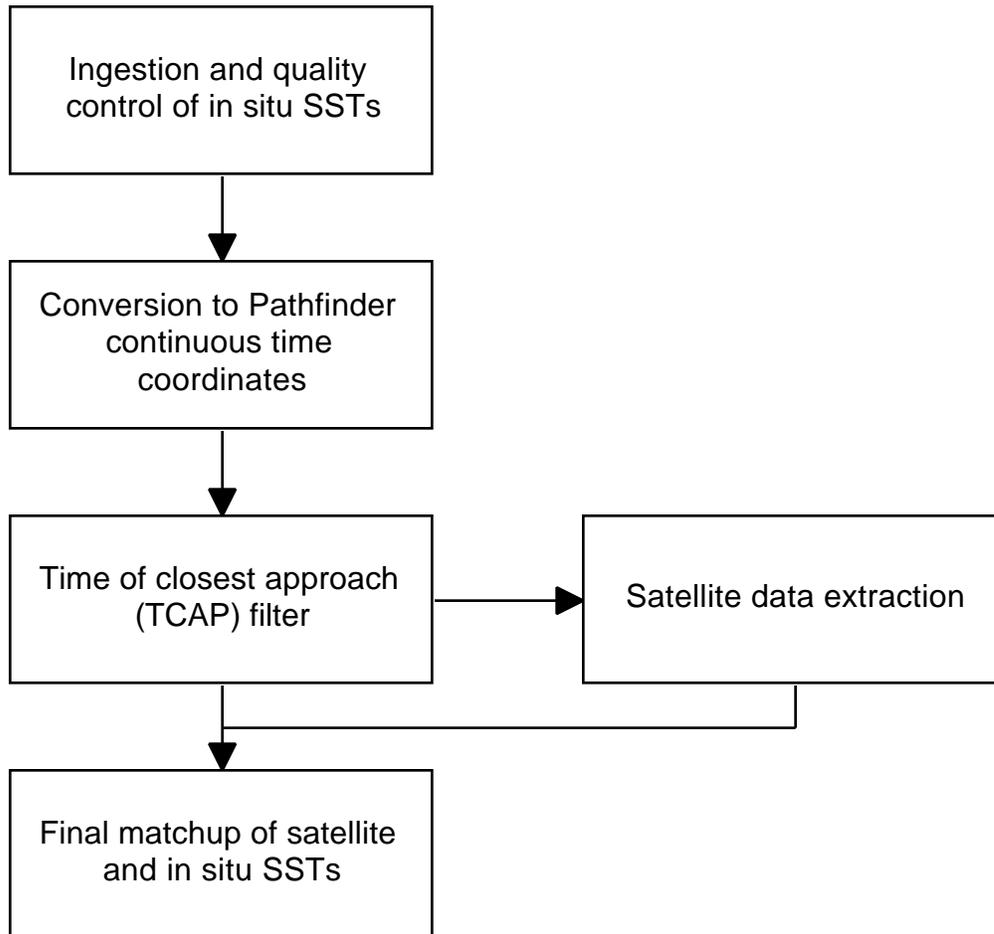


FIGURE 1. Major steps involved in the compilation of a matchup database. Emphasis is placed on the collection of *in situ* data. Details on the satellite data extraction process will be provided elsewhere.

The first step was the acquisition of *in situ* SST measurements from various sources. Our present effort is concentrated in the production of a matchup database for 1988. Nevertheless, the collection of environmental data was more encompassing, and we have compiled *in situ* SST data from November 1981 to the end of 1990. Two main types of *in situ* SST observations were used: data from moored or “fixed” buoys, and from drifting buoys. A brief description of the various data types is given below.

Moored Buoys:

NDBC Buoy: Moored buoys deployed by the US National Data Buoy Center (NDBC). Buoys are located off the Atlantic and Pacific coasts of the continental US, in the Gulf of Mexico, in the Gulf of Alaska, around the Hawaiian Islands and off the coast of Peru. The NDBC buoy data were provided by the National Oceanographic Data Center (NODC) and were pre-processed at NASA's Goddard Space Flight Center by Dr. Chuck McClain's research group.

Japanese Meteorological Agency Buoy: Data provided by the Japanese Meteorological Agency (JMA). Data correspond to a small number of buoys (3-4) around Japan.

TOGA/TAO Buoy: Data correspond to an array of moored buoys located in the Equatorial Pacific. Data were provided by NOAA's Pacific Environmental and Meteorological Laboratory (PMEL).

The moored buoy data from the various sources include different environmental variables. A list of the variables archived for each data source as part of the Pathfinder effort is shown in the following table. In some cases, variables other than the ones listed exist in the original data files, but they were considered of limited relevance for the Pathfinder objectives. For instance, a great deal of information is available for some of the NDBC buoys on wave spectra, but these data were not extracted from the original tapes.

*{SUBSCRIBER @EditionMgr @EditionClient @02EE20E8 \a * MERGEFORMAT}*

<i>Variable</i>	<i>NDBC</i>	<i>JMA</i>	<i>TOGA/TAO</i>
Sea surface temperature	✓	✓	✓
Air temperature	✓	✓	✓
Wet bulb temperature	✓	✓	
Wind speed	✓	✓	
Wind direction	✓	✓	
Significant wave height	✓	✓	
Wave period		✓	

Surface
pressure

atmospheric



Drifting Buoys:

AOML Drifting Buoys: This data set was provided by NOAA's Atlantic Oceanographic and Meteorological Laboratory (AOML). The drifting buoys are mostly distributed in the equatorial Pacific Ocean, although there are some in the Atlantic.

MEDS Drifting Buoys: This data set was originally obtained from Canada's Marine Environmental Data Service (MEDS) by Dr. McClain's group at NASA-GSFC. The data set includes all satellite-tracked buoys worldwide.

In both cases, the drifting buoy data included only latitude, longitude (derived from the ARGOS tracking system onboard the NOAA polar orbiters) and SST.

Problems with data sets

Even though the various *in situ* SST data sets were supposed to be fairly "clean" (i.e., no erroneous data or other problems because they were obtained from the primary archival locations for each data set), we determined that some problems existed. The problems originated either on the original data sets or during the reproduction of data. For instance, an unreadable data tape from MEDS resulted in a low number of *in situ* SST observations for the second half of 1988. This was easily apparent when listing the number of SST reports per month. In another case, some of the moored buoys in the TOGA/TAO array became "drifting buoys," as their location changed. Details on these location changes had to be obtained from NOAA/PMEL and corrected in the original data set. In some cases, the problems were fairly subtle and difficult to detect. An example of this is the discrepancies between drifter data in the MEDS and AOML data sets (detailed in a previous report), due to a change in the calibration of the SST sensors on some buoys, which was not corrected in the MEDS data.

The main lesson learned during the data compilation stage is that each data set has its own peculiarities. Therefore, there is always a need for quality checks, however simple. In our case, the quality

checks were limited to eliminating records without valid SST measurements, or records with SST values outside reasonable boundaries (-4 to 32°C). We made no attempt to check variables other than SST, or to perform more complicated checks on SST values (e.g., a spatial or temporal consistency check). This may be acceptable in this case, because quality control had already been performed by the archival sites. Nevertheless, the simple checks are always advisable.

The number of records with valid SST observations during 1988 is summarized in Table 2 for the moored buoys and Table 3 for the drifting buoys.

Conversion to Pathfinder time coordinates

The purpose of this step was to facilitate the temporal matchups by converting all the dates and times in the *in situ* records to a continuous time coordinate. The system coordinate chosen was “seconds since January 1, 1981” to accommodate the complete Pathfinder period. One aspect that may be considered is whether the fine temporal resolution (seconds) is really necessary, or if minutes may be acceptable.

Time of closest approach filter

The global matchup database requires that satellite data be extracted from the AVHRR/GAC passes for the times and locations for which we have *in situ* SST observations. Logistically, this requires that GAC data (archived as individual orbits) be restored to magnetic disk so that extractions could proceed. To increase the efficiency of the extraction process, we tried to avoid restoring passes for which no *in situ* SSTs are available. This required that we identify whether any *in situ* SSTs (from any of the data sources mentioned above) were included within the selected time/space matchup window for a given GAC pass. To accomplish this, we implemented a “time of closest approach” (TCAP) filter. All the *in situ* records were processed through the TCAP filter. If the location of a report was viewed by the AVHRR within the specified matchup time window around the time of the *in situ* observation, then the *in situ* SST passed the filter and a pointer to the appropriate AVHRR orbit was added to the record. The TCAP filter has two main advantages. First, the filter identifies which AVHRR orbits must be used in the extraction process, eliminating the need to restore every GAC orbit. Second, the

TCAP process eliminates a substantial portion of the *in situ* records, which are not viewed by the AVHRR. This reduces the amount of data to process.

Tables 2 and 3 show the number of *in situ* SST records that passed the TCAP filter. The tables show that the number of *in situ* observations decreases considerably once they have been filtered by the TCAP routine, which is to be expected. For instance, a fixed buoy which typically reports hourly data will be viewed by the AVHRR 2-3 times a day, so only 2 or 3 out of the 24 observations pass the filter for a given day. Data from the TOGA/TAO array are an exception. SST observations for all other data sources are instantaneous, whereas the values reported by the TOGA/TAO buoys are averaged over the reporting interval. For that reason, the matchup time window we chose for these data is considerably wider than the one used for the others (typically one hour). The consequence is that, if the matchup window is, say, plus or minus four hours, the chances of having an AVHRR orbit that views the area is much higher.

Final matchup of satellite and *in situ* SSTs

This part of the process is underway but has not yet been completed for the global GAC data. Nevertheless, all the methodologies involved, i.e., the satellite data extraction and the construction of the final matchup database, have been developed and tested by building the prototype US east coast and Gulf of Mexico matchup database described in previous reports. The prototype database not only served to test the matchup methodology from end to end, but it is also being actively used in the development of new SST algorithms. The prototype matchup database was built using only the NDBC moored buoys and AVHRR/HRPT data archived at the University of Miami.

B.4 DSP Support (H)

The following section provides summary of DSP support activities for 1992. These activities are performed primarily by Sue Walsh.

January 92

During January, there were numerous DSP activities that included testing, program modification, travel, and problem correction (fixes). The primary testing effort focused on REMAP to determine why combinations of input and output projections (and which specific combinations) fail near the date line. Three major modifications to the system were developed. First, the addition was the ability to ingest SeaSpace and Ocean Imaging raw data formats. Second, TCAP, a program to match a buoy location and data time with a satellite pass was added. Third, the ability to merge just part of graphics into an image was added. Sue spent two days at Goddard discussing DSP with the Pathfinder group.

A number of problems were corrected during January:

- Fixed PIXRD systems to output proper dim file format.
- Fixed FRNTEDG to output proper dim file format.
- Fixed TRACE to report input file open errors properly.
- Standardized the use of trig functions in the workstation source.
- Fixed MAKCAL to update the proper band of an image.

February, 92

During February, there were numerous DSP activities that included testing, program modification, and problem correction (fixes). DSP testing efforts included testing DSP's cursor handling on five different machines (DECstation, Silicon Graphics, Sun, VAXstation, and Adage). Additional testing was begun on SCRIPP ingester and SECTOR routine on Unix. DSP system modifications included checking the workstation DSP source into a repository managed by CVS; this is being done to all system routines as part of the source control effort. Additional capability was added to allow the use of NEWIMAGE to change the navigation information for an existing image.

A number of problems were corrected during February:

Fixed DFLOC in ORBIT, which fixed SECTOR.
Fixed DSP to update the proper header file for the modified image plane.
Fixed the ALIASSORT command procedure to work on Unix.

Problems corrected:

Fixed handling of calibration information in multi-band images.
Fixed ANLY command procedure to parse the filename properly on Unix.
Fixed MACE2 to make sure the cursor is within the data space.
Fixed some more command procedures to work on Unix.
Fixed COLORSHR to only access the CAL_INP directory when actually needed.
Fixed some problems with image plane (frame buffer) handling.
Fixed MIA2CDF on VMS.
Check into the source control some more of the VMS only source (e.g. WRKSPC).
Clean up many source files so they compile nicely on all systems.
Fix RLOCK to work on work on Unix.
Fixed WMEAN to check "logical" variables properly.
Fixed MAKE-BSD for Sun4.
Fix ANLY7D, CALEPS7, and MAPEPS7: La(n) from Lass(670) was wrong.
Fix the "get command line" routines.
Modify SSTBIN and SSTMOSAIC to use a different pixel quality algorithm; and
fixed the mean and standard deviation calculations (for linear calibration equations, it won't work for other calibration equations); bin the conc band instead of the ndvi band.

March, 92

During March, there were a number of DSP activities including testing, program modification, and problem correction (fixes). DSP testing efforts included more testing of SCRIPP and SECTOR on a DECstation. There were a significant number of modifications during March. Support was added to DSP for BSDI 386. MIA2TIFF was modified to only read the palette if there is an XFBD active. GRID was changed to output up to 60 lines instead of 40, and to print

labels along the whole width and height of an image instead of just 512 by 512. Further capability expansion included the ability to modify part of a palette, the ability for COMPOS to use a mask file, and modification of ANLY7D, CALEPS7 and MAPEPS7 to apply Wang coefficient to 670 La when using epsilons. A number of additional routines were added including: ANLY7D, CALEPS7, MAPEPS2, MAPEPS5, MAPEPS7, the MAKE-BSD utility, SSTBIN and SSTMOSAIC (bin sst data using the 18 km binning algorithm).

Problems corrected in March:

Fixed handling of calibration information in multi-band images.

Fixed the use of an error message routine.

Fixed MIA2TIFF to only read the palette if there is an XFBD active.

Fixed ANLY command procedure to parse the filename properly on Unix.

Fixed MACE2 to make sure the cursor is within the data space.

Fixed some more command procedures to work on Unix.

Fixed COLORSHR to only access the CAL_INP directory when actually needed.

Fixed some problems with image plane (frame buffer) handling.

Fixed MIA2CDF on VMS.

Check into the source control some more of the VMS only source (e.g. WRKSPC).

Clean up many source files so they compile nicely on all systems.

Fix RLOCK to work on work on Unix.

Fixed WMEAN to check "logical" variables properly.

Fixed MAKE-BSD for Sun4.

Fix ANLY7D, CALEPS7, and MAPEPS7: La(n) from Lass(670) was wrong.

Fix the "get command line" routines.

Modify SSTBIN and SSTMOSAIC to use a different pixel quality algorithm; and fixed the mean and standard deviation calculations (for linear calibration equations, it won't work for other calibration equations); bin the conc band instead of the ndvi band.

During April, DSP activities included testing, modifications, and problem resolution. Test efforts included testing the scripp ingester and sector on all 5 systems (DECstation, Silicon Graphics, Sun, VAXstation, and Adage). In addition, testing began for the new PATHSST atmospheric correction program, as well as PATHBIN, PATHTIME, and PATHMOS the new programs that bin sst data using 9 km bins. System modifications included the addition of an option

to have dsp menus displayed in a separate X window, a new version of the TABLE library that works on VMS, support for Motif on VMS and new geometry packers from Finland in the tk library.

Problems corrected:

- Changed MACE2 to only read the cursor location when necessary.
- Ensured the workstation ingesters to compile and link on VMS.
- Misc. fixes to XFBD for VMS and Unix.
- Install and fix usage of netcdf for VMS and Unix.
- Misc. fixes to libraries and utilities for VMS and Unix.
- Fix MIA2HDF for VMS.
- Misc. modifications to the ingesters for VMS.
- Misc. modifications to make ANSI compilers and Multinet happy.
- Fix CALLER for Sun.
- Fix EXIST for VMS.

May, 92

During May, DSP activities included testing, modifications, and problem resolution. Testing concentrated on additional examination of SCRIPP and SECTOR on the 5 machines and on testing of PATHSST, PATHBIN, PATHTIME, and PATHMOS on SGI. Modifications covered additions to PATHBIN, PATHTIME, PATHMOS that added associated data blocks to pass the grid size from the binner to mosaic, added mask and ch4mmm bands and added ascend, cloud, and sat zenith angle options to the command line. Further modifications included the ability to merge graphics with the PRINT XFBD command; programs to convert tbus data to ppt7 format, and to append new lines

to the satellite ephemeris files; support for VT300 terminals; support for multiple default directory paths on VMS; and, finally, the ability to read old style DSP image files on Unix.

Problems corrected:

- More misc. fixes to XFBD and libraries.
- Misc. fixes to DSP for VMS.
- Fixed array zeroing in stats.
- Misc. fixes to CALLER.
- Misc. changes to Ratfor.
- Fix error in usage of ESTANG result in COLORSHR and COLORSHR5.

June, 92

During June, DSP efforts included testing, modification and fixes. Testing continued on PATHSST, PATHBIN, PATHTIME, and PATHMOS. The SST equation in PATHSST was modified. PATHMOS has had mask1 and mask2 output bands added. PATHTIME was changed to create the output image, instead of requiring an existing image. edgemask was modified to work with any size/subsampled image. The library routine Dsp_IsPlane was added so programs can check input for file vs image plane. Finally, the UNWARN utility was added to support the VMS system.

Problems corrected:

- Fixed error messages in spacetime, and mosaic.
- Misc. modifications to CALLER.
- Fix use of merge in GSFCBIN.
- Palette fixes in XFBD.
- Fix navigation near the poles.
- Multiple changes to the PATH programs.
- Fixes to MAPEPS7 and ANLY7D.
- Correct handling of image plane header files.

B.5 Team Interactions

B.5.1 Spring Team Meeting (14-16 April 1992)

Bob Evans and Otis Brown participated in the April Team meeting; Otis acted as Ocean Team Leader. A 'straw-man' peer review proposal was prepared and used as a basis for MODIS Team model. During the meeting an Ocean Team position on Hughes-Santa Barbara proposed performance modifications for MODIS-N.

B.5.2 Ocean Team Meeting (20-22 May 1992)

Bob Evans and Otis Brown acted as joint hosts for the meeting in Miami. This meeting provided input into peer review and algorithm description documents. Discussions among Team members were held concerning MODIS-N performance trade-offs.

C. FUTURE ACTIVITIES

C.1 Database Future Work

C.1.1. Change the database language from RDML to SQL as Database Language

C.1.2. Change VMS system calls (lib\$delete, lib\$find_file, etc.) into operating-system independent RATFOR functions.

C.1.3. Change the interface itself from VMS-specific FORTRAN to operating-system independent RATFOR.

C.1.4. Further streamline and normalize the schema, particularly the MAIN and PROCESS_CONTROL tables.

C.1.5. Develop simple, easy to adapt, special Client/Server Functions

C.1.6. Change the Ingestor Function Control

C.1.7. Formalize Processing History Storage

C.1.8. Enter interface source, client/server source and associated database files into the CVS Source Control System

C.2 Client/Server Future Work

C.2.1. Creation of a resource manager and a performance monitor.

C.2.2. Expansion of the error handler to provide broader coverage and to integrate into the overall system error recovery scheme.

C.2.3. Cleanup of the errors which prevent unattended operation and test of the new code.

C.2.4. Continue testing the client/server with CZCS and AVHRR data. This would include the acquisition of a UNIX resident database to run parallel tests.

Short term tests would include additional runs with the complete CZCS framework modified such that the TIMEBIN and MOSAIC steps are run as separate jobs. This verification will be made using several additional 10 day selections of CZCS data.

C.3 Pathfinder (**P**)

C.3.1. Continue development of linking processes between *in-situ* and processed satellite data.

C.3.2. Expand the validation dataset.

C.4 Headquarters (**H**)

C.4.1. Create tools to assist in results interpolation.

C.4.2. DSP - Fix programs that access the graphics plane to use the navigation from the input image and not the graphics plane.

C.4.3 Refine PATH binning and mosaic pixel quality algorithm to eliminate clouds.

C.4.4 Verify workstation DSP (SGI, SUN, DECstation, VAXstation) by comparing each program's output with the Adage system.

C.5 Modis (**M**)

1. Hold discussions with terabyte storage system developers/vendors during the summer. Information obtained during the discussions will be available to interested parties.

D. PROBLEMS

D.1 Database Problems

None listed separately

D.2 Client/Server Problems

1. NSF Disks - There have been instances of NSF disks not being available when needed. This appears to be a problem at the system level rather than with this implementation.

D.3 Matchup Database Problems

None listed separately

D.4 DSP And Headquarters Related Problems

None listed.

APPENDIX A. PREVIOUS MATCHUP DATABASE SUBMISSION

The following section has been reproduced from the March 1992 Quarterly Report on contract NAS5-31362 as background material for Section B3.

III. DATA MATCHUP (P)

III.a NEAR TERM OBJECTIVES

Adding additional matchups to the database is clearly a near term objective. Examining ancillary datasets to determine their applicability within the matchup concept will continue.

III.b OVERVIEW OF CURRENT PROGRESS

A *MATCHUP* database includes various *in situ* and satellite quantities, coincident (or nearly coincident) in space and time. The *MATCHUP* database can be used to monitor sensor calibration and to test and improve geophysical algorithms. We have been developing a general methodology for doing matchups. There are four main steps:

- a. Compilation of *in situ* sea surface temperature (SST) and other environmental data.
- b. Development of a generic methodology for the identification of NOAA spacecraft orbits that coincide (within a specified time-space window) with *in situ* observations.
- c. Extraction of AVHRR data corresponding to times and locations where *in situ* data exist.
- d. Matchup of *in situ* and AVHRR data.

Step One involves compiling and reformatting the *in situ* data, primarily fixed and drifting buoy data. Reformatting is mainly a redating; it is easier to do matchups if the time coordinates are continuous. The date is converted into seconds relative to a reference data.

Step Two is the generation of a reduced list of *in situ* data. We have chosen to use orbit routines in *DSP* to generate a list of "times of closest approach" (*TCAP*). The *TCAP* methodology is generic in nature and can be used for any spacecraft, provided an orbital model is available. Its advantages are: (a) it reduces the volume of *in situ* data to consider further, (b) it provides a limited list of satellite orbits from which data are to be extracted.

Two checks are made; the first ensures that the observations fall within a specified temporal window and the second check ensures that the *in situ* location falls within the area scanned by the sensor. These steps produce a series of times, lats and lons of *in situ* observations, which will be fed to the extraction routines (the following step). This step excludes a significant amount of the *in situ* data.

Step Three extracts satellite data for the times and locations specified above. This step is sensor-specific and a decision must be made what data to extract. For example, using AVHRR, we extract data for a 3x3 pixel box centered at the *in situ* location for all 5 channels, plus geometric and sensor info (sat zenith angle, solar zenith angle, baseplate temperature).

Step Four merges the *in situ* and satellite extractions, building the *MATCHUP* data set. This step involves a further verification that the data fall within the specified time/space windows. The *MATCHUP* database can then be used for calibration and algorithm development and validation.

The experimental *MATCHUP* database is now being used to test new SST algorithms and criteria for cloud identification

III.c FUTURE ACTIVITIES

The *MATCHUP* database will be expanded and testing using the database will continue.

III.d PROBLEMS ENCOUNTERED

III.d.1 The GSFC group will no longer participate in the compilation of moored buoy data. We will need support to continue this activity (and we have already received some 1981 buoy data from NODC).

III.d.2 Similar to Problem 1, it involves lack of programming support for the compilation and processing of drifter data sets.

III.d.3 There was an error in our *TCAP* routines that results in lack of prediction for some locations.

Jim Brown has examined the prediction problem and, as a result, acquired a new set of *TCAP* routines from D. Baldwin (U. Colorado). After evaluating the implementation of the new *TCAP* program, it was determined that the improvement overcame many of the known problems. At this point, lists of global extractions can be generated.