

SANDIA REPORT

SAND2000-3011

Unlimited Release

Printed December 2000

ATR2000 Mercury/MPI Real-Time ATR System User's Guide

R. H. Meyer, D. W. Doerfler

Prepared By
Sandia National Laboratories
Albuquerque, NM 87185

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of
Energy under Contract DE-AC04-94AL85000

Approved for public release, further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (703) 605-6000
Web site: <http://www.ntis.gov/ordering.htm>

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed Copy: A03
Microfiche copy: A01



SAND2000-3011
Unlimited Release
Printed December 2000

ATR2000 Mercury/MPI Real-Time ATR System User's Guide

Richard H. Meyer, and Douglas W. Doerfler
Signal and Image Processing Systems Department
Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM 87185-0844

Abstract

The Air Force's Electronic Systems Center has funded Sandia National Laboratories to develop an Automatic Target Recognition (ATR) System for the Air Force's Joint STARS platform using Mercury Computer systems hardware. This report provides general theory on the internal operations of the Real-Time ATR system and provides some basic techniques that can be used to reconfigure the system and monitor its runtime operation. In addition, general information on how to interface an image formation processor and a human machine interface to the ATR is provided. This report is not meant to be a tutorial on the ATR algorithms.

Contents

Abstract.....	3
Introduction	5
Theory of Operation	5
Internode Communications	10
genericMCSMB	10
Booting and Startup	11
Getting to Know Your Way Around	18
Reconfiguration	19
Verbose Control Parameters.....	22
The Boot Process	24
References.....	25
Appendix A:.....	26
The “sysmc” command and PowerPC Task States	26
FOA Executive Process	26
FOA Process (currently only have FOA Executive)	26
INDX Executive Process	26
CDI Process (CEID4, CEID5, CEID6, CEID7, CEID8, and CEID9)	26
ID Executive Process	26
TMPM, TMSE, and PGA Process (CEID2 and CEID3)	27
TMPM, CPM and PGA Process (CEID11)	27
Appendix B:.....	28
Example: Mercury Startup Script File	28
Example: Mercury Configuration File.....	29
Example: ATR Startup Script File.....	31
Example: MPI Process Group File.....	32
Example: Executive Routing Files	33
Appendix C:.....	34
Interface Requirements	34
genericMyri	35
genericRPC (resavRPC)	36
Distribution:	37

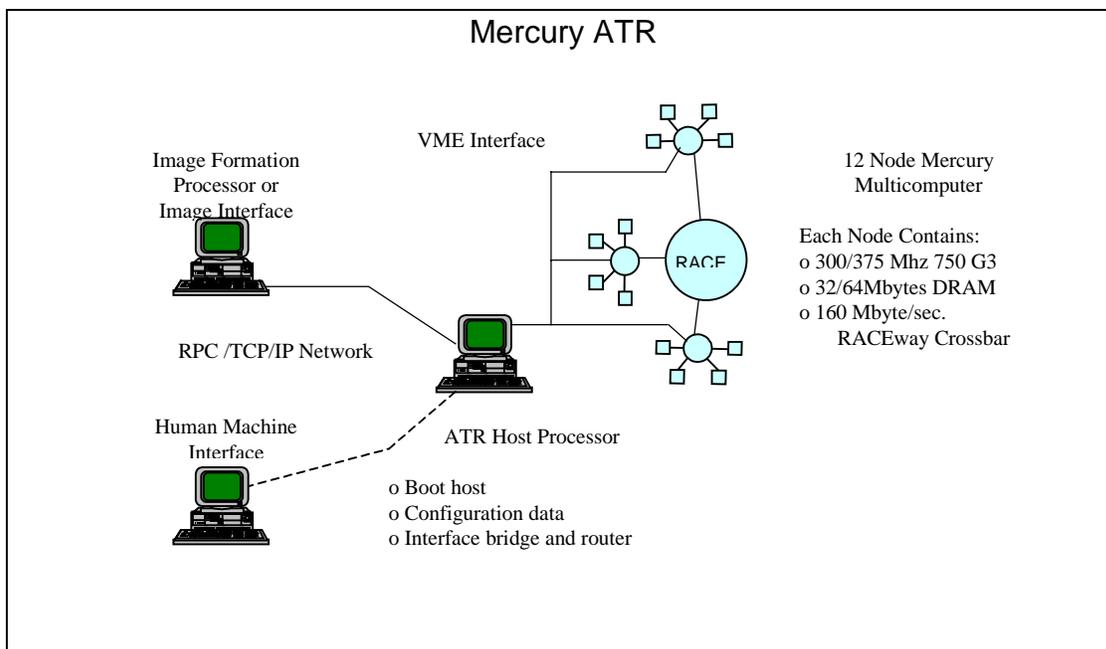
ATR2000 Mercury/MPI Real-Time ATR System User's Guide

Introduction

A real-time automatic target recognition (ATR) system has been developed for use with Mercury Computer systems hardware and operating system. The purpose of this document is to provide a user's guide to help personnel operating the ATR become more familiar with its theory of operation.

Theory of Operation

The ATR is a multiprocessing system and is based on COTS embedded high performance computing (EHPC) technology. The main ATR processing elements are 300Mhz and 375 MHz PowerPC 750 processors with 32 and 64Mbytes of memory respectively. Mercury⁵ RACEway crossbar switch technology is used for the interprocessor communications between the embedded nodes. There are 12 such nodes in the system.

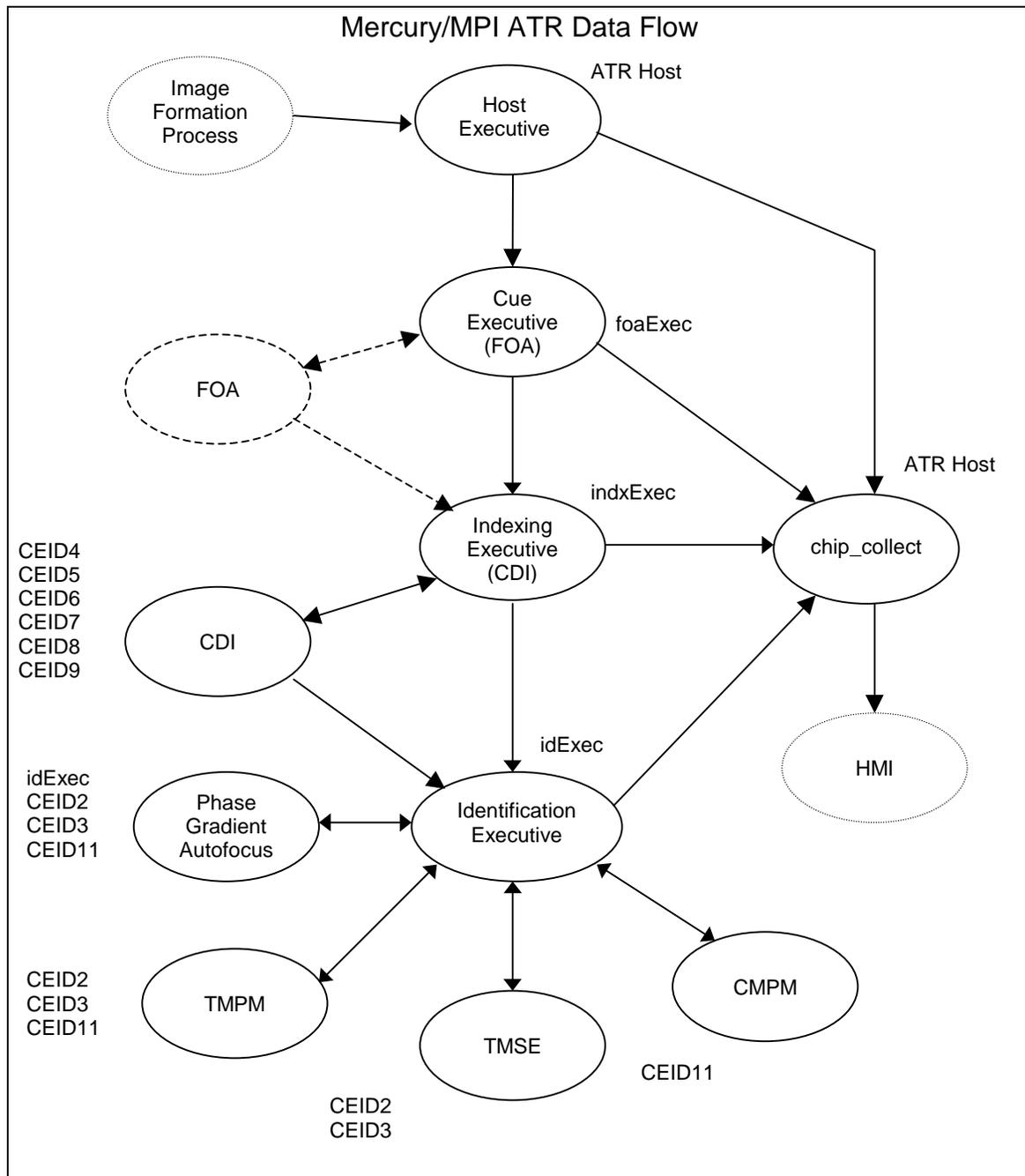


The system uses a Sun Microsystems workstation running the Solaris operating system for a host. The host node also contains a Mercury shared memory VME interface. This is the sole interface between the host and the compute nodes.

The host and compute nodes communicate with each other over the VME backplane using the Sandia ATR-Chip API and the Mercury Shared Memory Buffer (SMB) endpoints. The ATR-Chip API provides a software interface for nodes to represent and manipulate ATR data and symbolic information.

The Mercury RACEway interface provides node to node communications. It is implemented using the MPI Software Technology, Inc. (MTSI⁴) MPI/Pro™ Message Passing Interface⁶ software and the Sandia ATR-Chip API messaging protocol^{1,2}.

The ATR System implements several algorithms. A description of the algorithms is beyond the scope of this document. It is assumed the user is familiar with the basic theory of the ATR algorithms.



The system consists of the following components: The interface to SAR image formation process (IFP), the Image Former process (Host Executive), the Cueur Executive process, the FOA process(s), the INDXer Executive, the CDI process(s), the ID Executive process, and the TPM, TMSE, CMPM and Refocus processes.

The Host Executive (Image Former) process receives full SAR scenes from the IFP. Image Former buffers up the incoming data and when a full scene has

been acquired, it transmits the image to the FOA Executive. The Host Executive also has the ability to break very large scenes up into overlapping sub-scenes. The sub-scenes are overlapped so as not to cut a potential target into two pieces. If sub-scenes are formed they are transmitted individually to the FOA Executive node and are treated as separate images until the results are re-combined by the "chip_collect" process. At this time, duplicate ATR chip results are possible in the overlapping portion of the sub-scenes. The Image Former process also sends a message (i.e. number of sub-scenes) to tell the "chip_collect" process that an image has arrived and is being sent to the ATR.

The FOA Executive process receives scenes (or sub-scenes), breaks them into sub-images, and sends the sub-images to a specified number of nodes, which are executing the FOA process. Again, the sub-images are overlapped so as not to cut a potential target into two pieces. Note the FOA Executive executes the FOA algorithm and it sends the full scene to the INDX Executive. The FOA Executive also reports the number of sub-images it created to the "chip_collect" process.

Each FOA node performs the FOA algorithm on a sub-image and sends the FOA results to the INDX Executive. The number of nodes running the FOA process is determined by the expected workload of the system (i.e. input pixel rate) and is configured by the system architect. Each FOA node gets a sub-image and they operate in parallel.

The INDX Executive receives the SAR scene (or sub-scene) and ATR results from each of the FOA process(es). It buffers ATR results until the sub-scene components have been received. This buffering is due to the single-piped nature of the MPI communication model. It can cause a communication bottleneck and is an area that should be examined in future architectures. After all results for a sub-image have been received, the INDX Executive chips out the regions of interest as determined by FOA results; and routes the ATR chips to the CDI node(s). Again, the number of nodes running the CDI process is determined by the expected workload of the system (i.e. input pixel rate, image complexity, etc.) and is configured by the system architect. ATR chips are divided among available CDI nodes and they operate in parallel.

The ATR chips, which pass the CDI algorithm, are communicated to the ID Executive process.

ATR chips results that fail the CDI algorithm are sent, via the INDX Executive, to the "chip_collect" process on the ATR Host. Also, the total number of ATR chips created per image is sent to the "chip_collect" process.

In earlier implementations, the Cuer(FOA) and Indexer(SLD) processes were combined into a single process on the same processing nodes; and FOA/SLD ATR chips were routed directly to the MBV(ID) Executive process. However, the

current implementation separates the Cuer and Indexer functions on different nodes. The new INDX algorithm (CDI) is more complex and requires more processing power than the older Indexer (SLD).

The ID Executive receives the ATR results from each of the CDI process(s). It buffers ATR results until all CDI nodes have reported. This buffering is due to the single-piped nature of the MPI communication model. It can cause a communication bottleneck and is an area that should be examined in future architectures.

Once the ID Executive receives ATR chips from the CDI processes, it routes the chips to the identification (ID) algorithms TMPM, TMSE and CPM. Each chip does not necessarily get routed to each identification algorithm. The ID Executive performs some executive control logic which combines the results of all the ID algorithms into a single ID score, the TMD score. In order for a chip to be declared as containing a target, the combined TMD score of all algorithms must be lower than a given threshold. If at any point a chip's combined score grows greater than a given intermediate threshold, the target cue is declared a non-target and the cue is no longer processed and is forwarded to the ATR Host "chip_collect" process.

On the ATR Host, the "chip_collect" process queues the ATR chips for each SAR scene sent to the ATR system. ATR chips are buffered until all results for every scene are accounted for; then the entire chip list is transmitted to the HMI for operator viewing. Multiple SAR scene results may be queued on the "chip_collect" at any one time (currently a maximum of 5 scenes). Note the actual SAR image data is currently not stored in the "chip_collect" process; it must be transmitted directly from the SAR image formation processor (IFP) to the HMI.

The system architect, for a given set of conditions determines the number of nodes contained in an implementation of an ATR system. There can be multiple instances of any given process. All data flow for the system is specified in configuration files. An example description of configuration files can be found in Appendix B:.

Internode Communications

The Mercury ATR system uses a combination of communication protocols between the various system components (nodes). The external interface to the ATR implements the same interface requirements as previous ATRs (Joint STARS and TCTA - see Appendix C:). The Message Passing Interface (MPI) communication software was selected for internal ATR messaging for standardization purposes across multiple vendor platforms and ease of porting the ATR architecture to new platforms.

"genericRPC" is a server process used on the ATR Host to receive SAR scene data from external sources with RPC protocol. It passes the received scenes to the Host Executive Image Former process. This allows the ATR Host to immediately receive incoming data from a client (IFP), preventing the client from blocking and slowing other client tasks. It can be used in conjunction with the "genericMyri" server but this capability is not currently enabled on this version of the ATR system (see Appendix C: for more details.).

"resavRPC" is another server process used on the ATR Host to receive ATR results from internal clients with RPC protocol. It differs from the "genericRPC" server only in respect that it shares the input message queue(s) with the "genericMCSB" server. Host Executive Image Former ATR results are passed to the "chip_collect" process using this server.

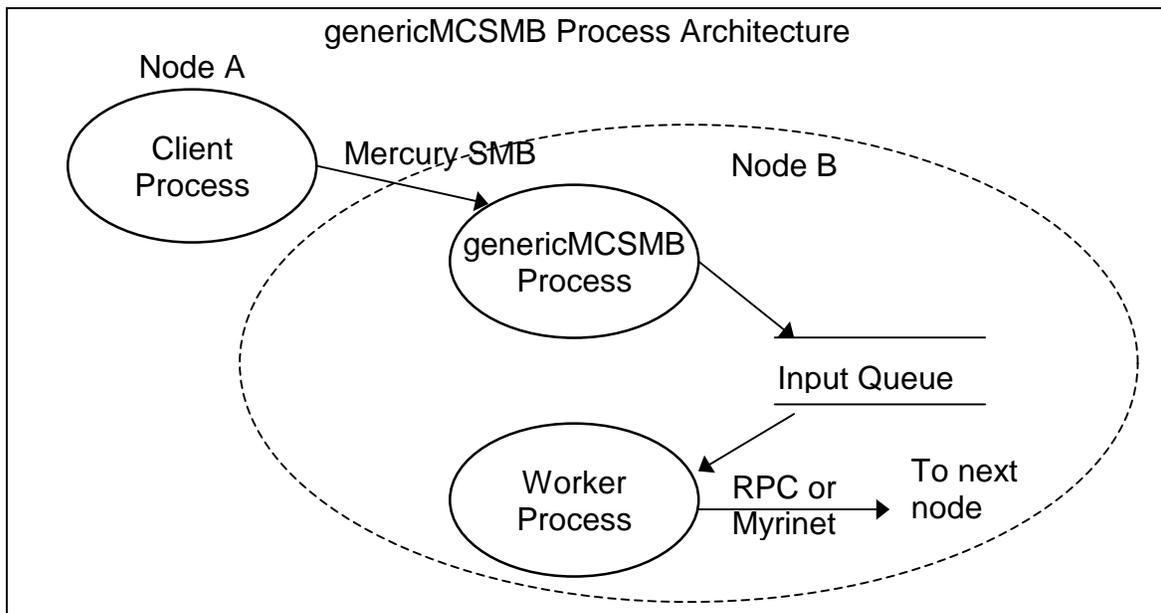
All internal messaging between Mercury Compute Environments (CEs), on the RACEway, is performed using the Message Passing Interface (MPI) communications software and the ATR Chip API.

The ATR Chip API is a software interface for managing ATR results and communicating imagery and results between processing nodes. The ATR Chip API is beyond the scope of this document. For details, refer to the documents "ATR Chip API User's Guide"¹ and "ATR Chip API Man Pages"². There are several Software protocols the ATR Chip API uses for sending and receiving data, Remote Procedure Calls (RPC), the Myrinet API, Mercury SMBs, and Message Passing Interface (MPI).

Since the MTSI MPI/Pro™ communication software for the Mercury operating system does not include the Solaris host, another host communication method was required. The Mercury Interprocessor Communications System (ICS) supports the use Shared Memory Buffers (SMBs) for this purpose. An ATR Chip API implementation which includes support for Mercury SMBs was implemented.

genericMCSMB

"GenericMCSMB" is a server process used on the ATR Host to receive image data and algorithm results from the internal Mercury CE sources using the Mercury Shared Memory Buffer (SMB) objects. A single instance runs on the ATR Host with Mercury SMB communications. The "genericMCSMB" process should run at a high priority. This allows the ATR Host to immediately receive incoming data from a client (FOAEXEC, INDEXEXEC, or IDEXEC), thus preventing the client from blocking and hence slowing other client tasks*. As messages are received from clients, they are put into a receiving buffer for a worker task, i.e. the "chip_collect" process. When a worker task is ready for a new ATR Chip message, it retrieves the message from the receive buffer. After processing the message, the updated ATR Chip information is sent to the next process in the data flow.



There can only be one "genericMCSB" task per node. If there are multiple worker tasks per node, the single "genericMCSMB" process receives all data and each worker process would have its own input queue. The client is able to indicate which buffer to put the data in by specifying a service number. At startup, "genericMCSMB" is configured with a given number of receive buffers and a service number for each buffer. The one major difference between "genericMCSMB" and the "genericMyri" process is that the "genericMCSMB" supports multiple channels (i.e. up to four SMB buffers).

Booting and Startup

* There is however a limit to the number of messages the genericMCSMB task will accept. If the limit is reached, it will no longer accept new messages. In this case the client process will be blocked. This prevents the host's memory from being used up if it is sent too many messages at one time.

The workstation serves as the host for the entire ATR system. It provides for initialization (booting CEs) and file system services. To bootstrap the entire ATR take the following steps:

1. Power up the ATR VME chassis and the host workstation.
2. Log into the ATR Host with the username "atruser" to ensure it has booted properly. Below is an example script. It may be desirable to set the DISPLAY environment variable.

```
$ rlogin mercury
*
*
*
login: atruser
Password: *****
*
*
*
mercury% set DISPLAY 199.26.46.29:0.0
mercury%
```

3. The Mercury Compute Environments (CEs) are not initialized or booted at this time. Two ATR scripts must be run in order to initialize the Mercury hardware and then load the individual CEs.

The first Mercury Startup script configures the installed Mercury VME boards and initializes the Configuration DataBase. The following example script will initialize the Mercury hardware configuration. This procedure must be executed once after the ATR Host has booted. An example Mercury Startup Script and Mercury Configuration File are included in Appendix B:.

```

*
mercury% cd /atrsys/atr99/bin/mcPPC
mercury% startup2
config init: parsing configuration file /usr/mercury/etc/atrsetup2.conf
*
          A rather large listing ( ~ 1 or 2 pages) will scroll by
*
Board MCH6_0 probed present
*
Board ILK4_1 probed present
*
CE host (CEID 1) probed present
CE CEID2 (CEID 2) probed present
CE CEID3 (CEID 3) probed present
CE CEID4 (CEID 4) probed present
CE CEID5 (CEID 5) probed present
CE CEID6 (CEID 6) probed present
CE CEID7 (CEID 7) probed present
CE CEID8 (CEID 8) probed present
CE CEID9 (CEID 9) probed present
CE FOAEXEC (CEID 10) probed present
CE CEID11 (CEID 11) probed present
CE INDXEXEC (CEID 12) probed present
CE IDEXEC (CEID 13) probed present
mercury%

```

4. A second ATR Startup script will automatically start the required processes on each Mercury CE. It takes a few minutes for the ATR to boot. Each PowerPC must go out to the VME backplane and load its operating system image, load all process executables, and initialize the MPI communication interface. Each process must read its configuration information and then load its appropriate configuration data. The following example script will initialize the Mercury operating system (MC/OS™) and start the Mercury ATR system. An example Mercury ATR Startup Script and MPI Process Group File are included in Appendix B:

```
*
mercury% cd /atrsys/atr99/bin/mcPPC
mercury% start_MCatr
[1] 10620
*
mpi_foaExec: Program Started
*
[NFS file access]
Config file: /atrsys/atrModules/templates/...
*
        A rather large listing ( ~ 4 or 5 pages) will scroll by
*
load mse templates done
*
load templates done
*
load mpm templates done
*
load CPM templates done
```

Note that the Mercury window prompt, on the ATR Host, never returns. This is because this window is now functioning as the console server for the Mercury ATR. Any error or standard I/O messages, for any of the Mercury CEs, will appear in this window.

To regain use of this window by typing the <Control C> character combination on the keyboard while this window is active. Note this aborts the Mercury CE name server CE. Then enter the following example to halt Mercury ATR system. Re-running the above script will restart it.

```
^Cmercury% halt_MCatr
killing 10620
*
        A bunch of Error messages (these can be ignored) will scroll by
*
FOAEXEC successfully reset
mercury%
```

Sometimes a Mercury CE cluster will not be reset by this 'halt_MCatr' script. If this should happen, it must be reset manually as follows.

```
mercury% sysmc -v -f FOAEXEC reset
*
*
FOAEXEC successfully reset
mercury%
```

Both of these scripts, 'startup2' and 'start_MCatr', could be executed by the ATR Host workstation at bootup. This would force an automatic Mercury ATR system boot by executing them from a '/etc/rc3.d/s99atr' file.

The best indication of the ATR state is to monitor the console server boot messages and wait for all configuration templates to be loaded. This also displays any error messages that may have occurred.

However, this console server window may not be available all the time (i. e. automatic booting). Another indication of when the ATR is ready for processing is to open a user window on the ATR Host workstation. Issue the MC/OS™ "sysmc" command to view the status of processes running on each CE. All algorithm processes will be in the READY state, except FOAEXEC, when it has finished booting. The "mpi_foaExec" process will be in the I/O BLOCKED state; waiting for the initial SAR scenes. You should see processes that have names like "mpi_foaExec", "mpi_indxExec", "mpi_indxExec.***", "mpi_idExec", and "mpi_id***...***". Appendix A: shows the state of CEs when they have successfully booted and are idle waiting for data to process. CMPM is usually the slowest to boot. If it is finished loading, it will be in the READY state. If it is still loading it's templates, it will be in the BLOCKED state.

```
% rlogin mercury
login: atruser
Password: *****
*
*
mercury% sysmc -v -f CEID11 ps
CE 11 (CEID11), Running, unreserved, name server 10, PPC with 64MB, on MB 3
PID      State  Flags  Block_id Args
0x000b0001  Ready  E
0x000b0021  Ready                mpi_idPgaMpmCMpm.ppc 0x20 -config /atrsys/...
Images: 0xb00000xb0001
*
*
mercury% exit
Connection closed.
```

Input SAR scenes for the ATR are received by the Host Executive Image Former process. Two processes on the host workstation must be started in order to accept SAR scenes. The process "hostExec" buffers the incoming data, breaks it into sub-images, and sends them to the FOA Executive. It uses "genericRPC" (and/or "genericMyri") to receive and buffer the input ATR images.

All output data from the ATR that is intended for the HMI is routed through the ATR Host workstation. Three processes on the workstation must be started in order to perform the routing. The process "chip_collect" queues up the ATR results on each SAR scene until all of it's component ATR chips have been processed and then performs the HMI routing function. It uses the "genericMCSMB" task to receive data from the ATR and the "resavRPC" task to receive sub-image information from the "hostExec" Image Former process.

These input and output processes are started by a script called "start_MCatr" in directory '/atrsys/atr99/bin/mcPPC'. This script may be executed automatically when the ATR workstation boots. The Solaris boot script '/etc/rc3.d/s99atr' is used to execute this script at boot time. If the host name of the HMI workstation changes, the "start_MCatr" script must be changed to reflect the new host name. To verify that each of the processes have started correctly, login to the ATR Host workstation and look for the following processes in the process list.

```

*
mercury% ps -u atruser
  PID  TTY    TIME  CMD
*
10622 pts/0  0:00  genericMCSMB
10630 pts/0  0:02  runmc
*
10621 pts/0  0:00  genericRPC
10628 pts/0  0:00  hostExec.mc
*
10562 pts/0  0:00  start_MCatr
10629 pts/0  0:00  chip_collect
10617 ?      0:00  sysmc
*
*
10620 pts/0  0:00  mc_fileServer
10626 pts/0  0:00  procMCSMB0
*
*
10623 pts/0  0:00  procMCSMB1
10627 pts/0  0:00  resavRPC
10625 pts/0  0:00  procMCSMB2
*
*
mercury%
*
```

At this point, the system is ready to receive images from the IFP.

Note that both the Image Formation Process (i.e. dhs) and the HMI Display process must be started independently. Each of these functions are separate operations from the ATR; and may consist of very diverse procedures on different hardware systems. A description of their operation is beyond the scope of this document but the interface requirements are included in Appendix C:.

Getting to Know Your Way Around

There are a total of 12 PowerPC processors comprising the Mercury ATR system. The PowerPC processors run the MC/OS™ real-time operating system, and the ATR workstation processor runs Sun Microsystems Solaris (a Unix variant) operating system. It is not necessary for the user to be familiar with these operating systems, but if the user wants to check the state of the processes and trouble shoot the system, basic knowledge of the Unix and Mercury Operating System MC/OS™ environment is required.

Unfortunately, the MC/OS™ operating system is not as flexible as some other real-time operating systems. In terms of allowing a user to examine internal process parameters from the command line, it is very limited. A source level debugger "gdbmc" is available for real-time operation; but it requires a high-level of knowledge of the OS (and '-g' re-compilation). The most helpful MC/OS™ console server command is the "sysmc .. ps" command which prints out all processes presently loaded. Using the "sysmc" command it is possible to check and make sure all processors have booted properly and that they have spawned all the appropriate tasks. Appendix A: provides example sessions and executing the "sysmc .. ps" command(s).

If the ATR is running correctly, it is not necessary to monitor its operation. If it is desirable to monitor the ATR a little more closely, the verbose parameter "-v" flag(s) may be set to some value in the CEs invocation string. This is a command line argument which is passed to each ATR process at it's invocation. Editing the verbose parameter '-v' in the Mercury ATR Startup Script (e.g. "start_MCatr") file and/or the MPI Process Group File (e.g. "jstarsATR.pg") will set it to the desired value.

By default (-v 0x20), the ATR Executive and Algorithm processes are configured to generate TIME results for each ATR chip and totals for each SAR scene. The ID Executive prints out these ATR TIME results for each SAR scene as it is processed.

The following example output shows the TIME results as displayed in the console server window. The exact meaning of each value in the printout varies from algorithm-to-algorithm and is beyond the scope of this document. It does represent the type of algorithms (FOA, CDI, etc.) which were utilized and the times for a SAR scene.

*							
TIME:							
alg	time1	time2	time3	arg1	arg2	arg3	
foa	0.500	0.500	0.500	100	2	0	
foa	0.250	1270000.000	0.000	1	1	0	
foa	0.250	1260000.000	0.000	0	0	0	
foa	0.500	0.500	0.500	200	1	0	
cdi	0.200	75000.000	0.000	2	2	4	
*							
cdi	0.340	0.340	0.340	300	7	0	
mpm	0.070	240000.000	0.000	1	1	3	
*							
pga	0.070	230000.000	0.000	0	0	0	
mse	0.280	57000.000	0.000	5	5	2	
*							
cmpm	0.120	140000.000	0.000	5	5	3	
?	0.570	0.570	0.570	400	3	0	
*							

Reconfiguration

In theory, any algorithm process can be run on any CE node. The purpose of this section is to show the user how to reconfigure a given CE node to run a given algorithm. This may be helpful if a node fails due to a hardware problem and needs to be removed from the system.

Since the Mercury ATR system comprises PowerPC's with two memory sizes, 32 MBytes and 64 MBytes; some care should be exercised when selecting node functions. In general, larger memory sizes should be reserved for FOA processes and Executive processes. This is because the large SAR scenes and ATR chip images are stored in process heap memory. It would not work if one tried to allocate 50 MBytes of heap memory on a 32 Mbyte CE node.

The algorithm processes that a given CE node executes are determined by which ATR Startup script "start_MCatr" and the MPI Process Group file "jstarsATR.pg" which are executed to start the ATR. Startup scripts and their corresponding Process Group files are located in the directory '/atrsys/atr99/bin/mcPPC' on the host workstation.

Two other files that may have an influence on the reconfiguration process are the Mercury Startup Script "startup2" and Mercury Configuration File "atrsetup2.conf". These two files determine the CE node names which are assigned to each PowerPC (i. e. FOAEXEC, INDXEXEC, IDEXEC, etc.). Be careful if editing the "atrsetup2.conf" configuration file; the file format is unique for the Mercury hardware setup program. Only change the "CE node names" unless very familiar with the Mercury OS. The Mercury Startup script "startup2" is located in the directory '/atrsys/atr99/bin/mcPPC' on the host workstation. The Mercury Configuration File "atrsetup2.conf" is located in the directory

'/usr/mercury/etc' and you may need 'super-user' access' privileges in order to change it.

If the Mercury Startup script "startup2" and/or Mercury Configuration File "atrsetup2.conf" are modified, the configuration data base must be reset as follows. For an example of the Mercury Startup script and Mercury Configuration file see Appendix B:.

```
mercury% sysmc -v -f FOAEXEC reset
*
      { resets all CE nodes - FOAEXEC is the name server }
FOAEXEC successfully reset

mercury% configmc -v reset
*
      { Mercury Configuration DataBase reset }
Mercury Configuration DataBase successfully reset

mercury% startup2
config init: parsing configuration file /usr/mercury/etc/atrsetup2.conf
*
      A rather large listing ( ~ 1 or 2 pages) will scroll by
*
Board MCH6_0 probed present
*
Board ILK4_1 probed present
*
CE host (CEID 1) probed present
CE CEID2 (CEID 2) probed present
CE CEID3 (CEID 3) probed present
CE CEID4 (CEID 4) probed present
CE CEID5 (CEID 5) probed present
CE CEID6 (CEID 6) probed present
CE CEID7 (CEID 7) probed present
CE CEID8 (CEID 8) probed present
CE CEID9 (CEID 9) probed present
CE CEID10 (CEID 10) probed present
CE FOAEXEC (CEID 11) probed present
CE INDEXEXEC (CEID 12) probed present
CE IDEXEC (CEID 13) probed present
mercury%
***** Note Change *****
***** Note Change *****
```

In previous example, the CE 10 & CE 11 nodes, both 64 MBytes of memory, were renamed. E.g., let's change the role of CE node11 to run the FOA Executive and CE node10 to run the ID algorithms.

Note the ATR Startup script "start_MCatr" requires one change because CE nodes are referred to by name (e.g. FOAEXEC, INDEXEXEC, etc.). Use any text editor (e.g. vi, ..) to make changes. For an example ATR Startup script and MPI Process Group file see Appendix B:.

```
*
mercury% cd /atrsys/atr99/bin/mcPPC
mercury% vi start_MCatr

    { change the following line - even this could be avoided if
      I'd used CEID10 in the first place for FOAEXEC in this line only
      but this is more readable ? }
systemc -f FOAEXEC -bcs=0 init FOAEXEC CEID10

:wq
mercury%
```

Next one change is required in the MPI Process Group File "jstarsATR.pg". Use any text editor (e.g. vi, ..) to make changes.

```
*
mercury% cd /atrsys/atr99/bin/mcPPC
mercury% vi jstarsATR.pg
    { change the following line - last line in file }
CEID 10 mpi_idPgaMpmCMpm.ppc -h 52428800 -v 0x20 ....
*
:wq
mercury%
```

Finally three changes must be made to the routing file "id_jstars.cfg". An example routing configuration file for the ID Executive is given in Appendix B:.

```
*
mercury% cd /atrsys/atr99/config/robust
mercury% vi id_jstars.cfg
    { change the following lines - CEID10 }
route -alg mpm -host CEID10 -max 1 -tfm LOG -fmt UINT16 ....
*
route -alg cmpm -host CEID10 -max 1 -tfm LOG -fmt UINT16 ....
*
route -alg pga -host CEID10 -max 1 -tfm IQ -fmt SCOMPLEX ....
*
:wq
mercury%
```

Restart the Mercury ATR by running the ATR Startup script "start_MCatr" as before.

Note this reconfiguration example was one of the most difficult possible because it involved PowerPC module name changes. However, it did take us through all the possible reconfiguration steps. Many CE node changes require only changes to the MPI Process Group file "jstarsATR.pg" and/or the routing file "id_jstars.cfg".

Verbose Control Parameters

In terms of allowing a user to examine internal process parameters from the command line, the MC/OS™ operating system is very limited. A source level debugger "gdbmc" is available for real-time operation; but it requires a high-level of knowledge of the OS ('-g' re-compilation) and may alter the program flow. The previously mentioned "sysmc" command (see Appendix A:) is the only available runtime command.

Unlike previous ATR versions, individual process parameters are not available from the VxWorks shell command line. To compensate for this lack of internal parameter visibility, an extensive set of verbose (-v) option flag settings are included within the process modules. These verbose flags are set at program initialization (boot time) by the ATR Startup script "start_MCatr" and the MPI Process Group file "jstarsATR.pg" (see Appendix B:). They are intended for debugging purposes and not as general user commands.

The core of the ATR system is the FOA, INDX, and ID Executives through with all of the ATR images must pass. This Executive(s) software takes care of routing chips to nodes. The difference between the FOA/INDX Executive(s) and ID Executive is in the scoring software that determines routing for the ID Executive. The FOA and INDX Executive have simple routing scheme(s), but the ID Executive is more complicated because it has to combine the results of the various identification stage algorithms. These Executive nodes are the first to places to examine when trying to understand or trouble-shoot ATR problems.

When a certain verbose (-v 0x01) flag is set, information on a specific topic is displayed in the console server window for that CE node. A high-level knowledge of internal ATR process operation may be required to understand this printout. This verbose capability is here for anyone who may have a more in-depth understanding of the ATR operations.

Each CE node in the Mercury ATR system has it's own verbose flag and the various options may be bit-wise OR'ed together. Because all the verbose information is displayed on one console server window, the verbose flag for individual CEs are usually set singularly (or in pairs). This prevents a flood of incomprehensible information scrolling across the screen from separate CE nodes.

The meaning of each bit set in the verbose flag word varies from process-to-process and is beyond the scope of this document. Generally, three types of verbose flag bit settings are useful to the ATR system manager and are explained in the following table.

Verbose Bit (Hex)	Function	Description
0x00000001	Display Progress	Prints out Executive Progress as ATR image passes through CE node.
0x00000020	TIME	Attach ATR Timing results to ATR chip list. Display Timing results after ID Executive completes.
0x00001000	Display ATR Chip List "atr_chipShow"	Prints out ATR chip lists at selected points in the Executive flow. <i>(Note: This may be a large list).</i>

Many more Executive verbose flag bit(s) are available but they may have meaning only for individual ATR Executive and algorithm processes. Consult, individual executive, communication, or algorithm source(s) for detailed operation and meanings.

Several algorithm specific verbose flags may be set in the ATR Startup script "start_MCatr" and the MPI Process Group files "jstarsATR.pg". These include the following configuration parameters. Check the individual algorithm documentation (sources) for values and uses.

Configuration Parameter	Algorithm Verbose Flag	Description
-foav lev	microfoa_verbose	FOA Algorithm Flag Level
-cv lev	microcdi_verbose	CDI Algorithm Flag Level
-tmd_verbose lev	tmd_verbose	TMD Scorer Flag Level
-pgav lev	pga_verbose	PGA Algorithm Flag Level
-msev lev	micromse_verbose	MSE Algorithm Flag Level
-mpmv lev	mocrompm_verbose	MPM Algorithm Flag Level
-cmpmv lev	microcmpm_verbose	CMPM Algorithm Flag Level

The Boot Process

The most critical procedure for the ATR is the boot process. If a node fails, it most likely will occur at boot and/or program initialization.

The best indication of the ATR state is to monitor the console server boot messages and wait for all configuration templates to be loaded. This also displays any error messages that may have occurred.

At power-up all the Mercury Compute Environment (CE) VME boards will display two yellow LEDs (PRC1 and PRC2) as the normal reset mode.

As the compute nodes are initialized (i.e. "sysmc ... init"), the yellow LEDs extinguish and green LEDs (processors A, B, C, and D) flash depending on which CE processor is being loaded. Also, the yellow (orange) VME LED indicates VME backplane activity. When all CEs are loaded and initialized, the green front panel LEDs will extinguish (turn OFF) except for bursts of monitoring activity.

At this point, the green LEDs are functioning as an activity indicator for each CE node. A special CE processor (e.g. currently CE 10) is designated as the Mercury name server node and may show more activity than the other nodes. During periods of SAR scene downloading, the FOAEXEC (i.e. CE 10) will indicate significant VME bus activity.

If an error is detected or a required process(s) dies, the Error message will be displayed in the Console Server window.

Pressing the "Reset" button on the front panel of the any Mercury VME board will reset the individual board. However, if any board is reset the whole ATR system will need to be halted and re-initialized (see Booting and Startup, section 4).

References

- ¹D. W. Doerfler, *ATR Chip API User's Manual*, internal documentation in HTML format, Sandia National Laboratories, Albuquerque, NM, version 10/00.
- ²D. W. Doerfler, *ATR Chip API User's Man Pages*, internal documentation in HTML format, Sandia National Laboratories, Albuquerque, NM, version 11/00.
- ³<http://www.myri.com>, Myricom, Inc., Arcadia, CA.
- ⁴<http://www.mpi-softtech.com>, MPI Software Technology, Inc., Starkville, MS
- ⁵<http://www.mc.com>, Mercury Computer Systems, Inc., Chelmsford, MA
- ⁶M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, *MPI: The Complete Reference*, The MIT Press, Cambridge, Massachusetts, 1996.

Appendix A: The “sysmc” command and PowerPC Task States

FOA Executive Process

```
mercury% sysmc -v -f FOAEXEC ps

CE 10 (FOAEXEC), Running, unreserved, name server 10, PPC with 64MB, on MB 3
  PID      State   Flags  Block_id Args
  0x000a0001 Ready   E
  0x000a0021 Blocked                d45a0 mpi_foaExec -max_bytes 1000000 -co -1 ...
Images: 0xa0000 0xa0001

mercury%
```

FOA Process (currently only have FOA Executive)

```
mercury% sysmc -v -f CEID?? ps

CE ?? (CEID??), Running, unreserved, name server 10, PPC with ??MB, on MB ?
  PID      State   Flags  Block_id Args
  0x000?0001 Ready   E
  0x000?0021 Ready                mpi_foaExec.ppc 0x20 -co -1 -config /...
Images: 0x?0000 0x?0001

mercury%
```

INDX Executive Process

```
mercury% sysmc -v -f INDXEXEC ps

CE 12 (INDXEXEC), Running, unreserved, name server 10, PPC with 64MB, on MB 3
  PID      State   Flags  Block_id Args
  0x000c0001 Ready   E
  0x000c0021 Ready                mpi_indxExec 0x20 -config /atrsys/...
Images: 0xc0000 0xc0001

mercury%
```

CDI Process (CEID4, CEID5, CEID6, CEID7, CEID8, and CEID9)

```
mercury% sysmc -v -f CEID4 ps

CE 4 (CEID4), Running, unreserved, name server 10, PPC with 32MB, on MB 1
  PID      State   Flags  Block_id Args
  0x00040001 Ready   E
  0x00040021 Ready                mpi_indxExec.ppcdi 0x20 -config /...
Images: 0x40000 0x40001

mercury%
```

ID Executive Process

```
mercury% sysmc -v -f IDEXEC ps

CE 13 (IDEXEC), Running, unreserved, name server 10, PPC with 64MB, on MB 3
```

```
PID      State  Flags  Block_id  Args
0x000d0001  Ready  E
0x000d0021  Ready
Images: 0xd0000 0xd0001
mpi_idExec 0x20 -indexer cdi -config /...
```

mercury%

TMPM, TMSE, and PGA Process (CEID2 and CEID3)

mercury% sysmc -v -f CEID2 ps

```
CE 2 (CEID2), Running, unreserved, name server 10, PPC with 32MB, on MB 1
PID      State  Flags  Block_id  Args
0x00020001  Ready  E
0x00020021  Ready
Images: 0x20000 0x20001
mpi_idPgaMpmMse.ppc 0x20 -config /...
```

mercury%

TMPM, CPM and PGA Process (CEID11)

mercury% sysmc -v -f CEID11 ps

```
CE 11 (CEID11), Running, unreserved, name server 10, PPC with 64MB, on MB 3
PID      State  Flags  Block_id  Args
0x000b0001  Ready  E
0x000b0021  Ready
Images: 0xb0000 0xb0001
mpi_idPgaMpmCMpm.ppc 0x20 -config /...
```

mercury%

Appendix B:

Example: Mercury Startup Script File

/atrsys/atr99/bin/mcPPC/startup2 :

```
# Mercury Startup Script
#
# Note: This script must be run once after the Solaris ATR Host is booted.

# Initialize the MCOS Configuration Data Base.
setenv MC_show_progress
configmc -cf /usr/mercury/etc/atrsetup2.conf init

# Set Environmental Variables.
setenv MC_device FOAEXEC
setenv MC_name_server FOAEXEC
setenv MC_become_console_server
setenv MC_exec_heap_size 64K
setenv MC_heap_size 20M
setenv MC_grm_size 32K
# unsetenv MC_show_progress

# Initialize(Boot) the Target Cluster(s).
# sysmc -f FOAEXEC init CEID10 CEID11
# sysmc -f CEID2 -ns FOAEXEC init CEID2 CEID3 CEID4 CEID5
# sysmc -f CEID6 -ns FOAEXEC init CEID6 CEID7 CEID8 CEID9
# sysmc -f CEID12 -ns FOAEXEC init CEID12 CEID13

# Note: to Reset the MCOS Configuration Data Base.
# This script (or another like it) must be run if the following command is used !
# configmc -v reset

# To Load/Run a VxWorks Target Image(s) - not used with Solaris Host.
# setenv MC_remote_host atr_host
# rspmc - &
```

Example: Mercury Configuration File

/usr/mercury/etc/atrsetup2.conf :

```
# Mercury Config with Force CPU-50GT Ultra-Sparc Ili Board in Slot#1.
#
# Mercury config file for three MCH6 boards all fully populated (P2F);
# with four PPC's. The three boards are connected by an ILK4
# interlink module.
#
# nodes on the MCH6 are PowerPC
#
# For details on ppc board defintions refer to the ppc.conf examples

# PowerPC 6U board with 4 processors
board MCH6 -name MCH6_0
port VME -pVME 0x10000000 -pVMEws 32M -VMEil 3 -VMEiv 0xfb
port ILK -num 0
node A -memsize 32M -ptesize 0x100000 -node_name "CEID2"
node B -memsize 32M -ptesize 0x100000 -node_name "CEID3"
node C -memsize 32M -ptesize 0x100000 -node_name "CEID4"
node D -memsize 32M -ptesize 0x100000 -node_name "CEID5"

# PowerPC 6U board with 4 processors
board MCH6 -name MCH6_1
port VME -pVME 0x12000000 -pVMEws 32M -VMEil 3 -VMEiv 0xfa
port ILK -num 0
node A -memsize 32M -ptesize 0x100000 -node_name "CEID6"
node B -memsize 32M -ptesize 0x100000 -node_name "CEID7"
node C -memsize 32M -ptesize 0x100000 -node_name "CEID8"
node D -memsize 32M -ptesize 0x100000 -node_name "CEID9"

# PowerPC 6U board with 4 processors
board MCH6 -name MCH6_2
port VME -pVME 0x14000000 -pVMEws 64M -VMEil 3 -VMEiv 0xf9
port ILK -num 0
node A -memsize 64M -ptesize 0x100000 -node_name "FOAEXEC CEID10"
node B -memsize 64M -ptesize 0x100000 -node_name "CEID11"
node C -memsize 64M -ptesize 0x100000 -node_name "INDEXEXEC CEID12"
node D -memsize 64M -ptesize 0x100000 -node_name "IDEXEC MBVEXEC CEID13"
```

```

#####
##
## FORCE 50GT Solaris 2.6.1
##
## Sample configuration for host
## use host_vme_register ("HOST_DEV", 0x08000000, 0x2000000)
##
#####
board FRC50GT -board_name HOST_BRD -probe=0
port VME -VME 0x08000000 -VMEws 32M
device A -device_name HOST_DEV -memsize 32M
#####"

# Defintion for four-slot motherboard interconnect ILK4. The ILK4
# board attaches to the P2 connector of the VME backplane and provides
# raceway connections between the four motherboards in the system.
board ILK4 -board_name ILK4_1
# ILK ports 0 - 3 are raceway connections on the backplane
# numbered left to right in a horizontal chassis
port ILK -num 0
port ILK -num 1
port ILK -num 2
port ILK -num 3
# ILK ports 4 and 5 allow this ilk to be connected to other ILK boards
port ILK -num 4 # port 4 daisy-chain directive
port ILK -num 5 # port 5 daisy-chain directive

# The MCH6 board is in slot 0 of the ILK4
connect ILK ILK4_1 0 MCH6_0
# The MCH6 board is in slot 1 of the ILK4
connect ILK ILK4_1 1 MCH6_1
# The MCH6 board is in slot 2 of the ILK4
connect ILK ILK4_1 2 MCH6_2
# The MCH6 board is in slot 3 of the ILK4
# connect ILK ILK4_1 3 MCH6_3

```

Example: ATR Startup Script File

/atrsys/atr99/bin/mcPPC/start_MCatr :

```
#!/usr/bin/csh
kill_MCatr
#set Default Heap size to 20 Mbytes.
setenv MC_heap_size=0x1400000
#set Default Stack Size to 128 Kbytes.
setenv MC_stack_size=0x00020000
#set Default GRM Size to 64 Kbytes.
setenv MC_grm_size=0x00010000
# Initialize MCOS nodes; Name Server - FOAEXEC
sysmc -f FOAEXEC -bcs=0 init FOAEXEC CEID11
sysmc -f INDXEXEC -ns FOAEXEC -bcs=0 init INDXEXEC CEID4 CEID5 CEID6 CEID7 CEID8 CEID9
sysmc -f IDEXEC -ns FOAEXEC -bcs=0 init IDEXEC CEID2 CEID3
mc_fileServer &
# start Communication Process(s) (for hostExec).
#genericMyri -qsize 4 -channel 2 -ckey 1022 -rkey 1023 &
genericRPC -qsize 4 -prog 0x20000010 -ckey 1022 -rkey 1023 &
# start Communication Process(s) (for chip_collect).
genericMCSMB -qsize 4 -ce 10 -smb 0 -ce1 12 -smb1 -ce2 13 -smb2 -prog 20000009 \
    -ckey 1016 -rkey 1017 &
sleep 1
resavRPC -qsize 4 -prog 0x20001000 -ckey 1016 -rkey 1017 &
# start Host Executive (image Former).
# Max - Scene Size - 24 Mbytes; ATR Report Host - FOAEXEC; Display Host (chip_collect)- atr_host.
hostExec.mc -ch FOAEXEC -np 12 -tfm IQ -fmt scomplex -chunk_size 0x1800000 \
    -dh atr_host -dp 0x20001000 &
# start Display Interface Program.
# Chip List Queing(buffered); Verbose - Off; HMI Host - hmi_host.
chip_collect -v 0x00 -nh hmi_host -np 20000009 &
# Chip List (un-buffered); Verbose - Off; HMI Host - hmi_host.
#resav -nh hmi_host -np 20000009 &
# start Mercury/MPI ATR system.
runmc -h 50M -f FOAEXEC mpi_foaExec.ppc -rank 0 -size 12 -pg_file ./jstarsATR.pg \
    -max_bytes 1000000 -co -1 -v 0x20 -rh INDXEXEC -dh atr_host -hostrank 0 \
    -config /atrsys/atr99/config/robust/algorithm.cfg \
    -confoa /atrsys/atr99/config/robust/foa.cfg
sleep 1
# kill all MCOS processes executing on Mercury CE's.
kill_MCatr
# reset MCOS CE's.
sysmc -f FOAEXEC reset
sysmc -f INDXEXEC reset
sysmc -f IDEXEC reset
```

Example: MPI Process Group File

/atrsys/atr99/bin/mcPPC/jstarsATR.pg :

```
#-----  
# This is the jStars ATR ProcGroup file... the format is  
#  
# CENAME      /path/to/execname1.ext [args...]  
# CEID        5      /path/to/execname2.ext [args...]  
# CEIDs       6-10  /path/to/execname3.ext [args...]  
# .           .           .  
# .           .           .  
# .           .           .  
#  
# Where ext is ppc.  
# Ranks assigned to machines in order that they are listed.  
#  
#-----  
# here are some PPC's  
FOAEXEC      mpi_foaExec.ppc -h 52428800 -v 0x20 -co -1 -max_bytes 1000000 \  
-config /atrsys/atr99/config/robust/algorithm.cfg \  
-confoa /atrsys/atr99/config/robust/foa.cfg -rh INDEXEXEC  
INDEXEXEC    mpi_indxExec.ppcdi -h 52428800 -v 0x20 \  
-config /atrsys/atr99/config/robust/algorithm.cfg -rh IDEXEC -dh atr_host \  
-hostrank 1  
IDEXEC       mpi_idExec.ppc -h 52428800 -v 0x20 -indexer cdi \  
-config /atrsys/atr99/config/robust/algorithm.cfg \  
-conid /atrsys/atr99/config/robust/id_jstars.cfg -rh atr_host -hostrank 2 \  
-pgatype mpm  
CEIDs 2-3    mpi_idPgaMpmMse.ppc -h 20971521 -v 0x20 \  
-config /atrsys/atr99/config/robust/algorithm.cfg \  
-conid /atrsys/atr99/config/robust/id_jstars.cfg  
CEIDs 4-9    mpi_indxExec.ppcdi -h 20971521 -v 0x20 \  
-config /atrsys/atr99/config/robust/algorithm.cfg -rh IDEXEC \  
CEID 11      mpi_idPgaMpmCMpm.ppc -h 52428800 -v 0x20 \  
-config /atrsys/atr99/config/robust/algorithm.cfg \  
-conid /atrsys/atr99/config/robust/id_jstars.cfg
```

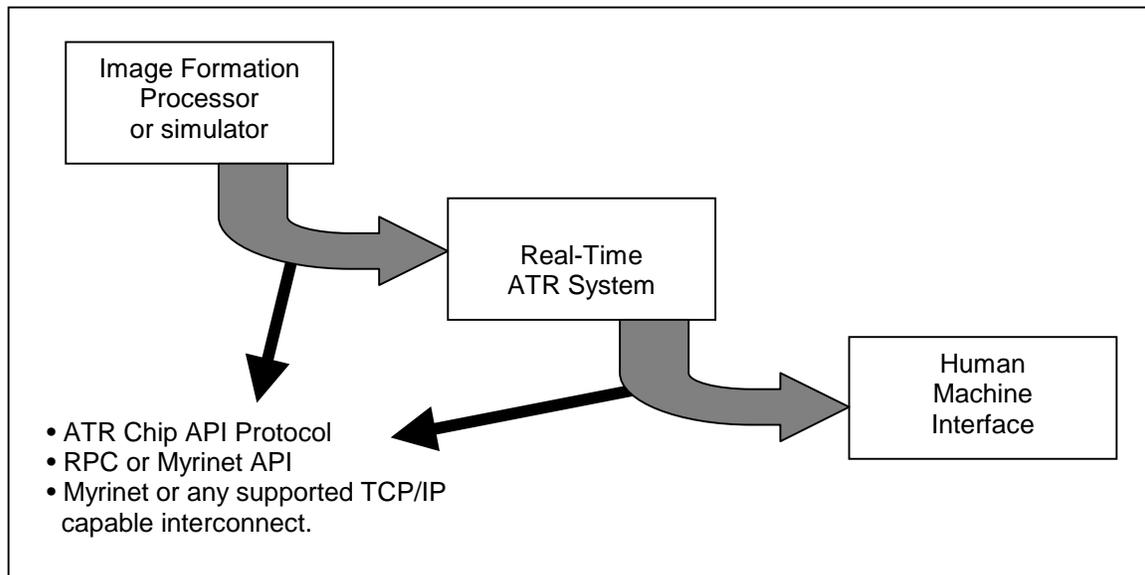
Example: Executive Routing Files

/atrsys/atr99/config/robust/id_jstars.cfg :

```
route -alg mpm -host CEID2 -max 1 -tfm LOG -fmt UINT16 -smp1X -pause 15
route -alg mpm -host CEID3 -max 1 -tfm LOG -fmt UINT16 -smp 1X -pause 15
route -alg mpm -host CEID11-max 1 -tfm LOG -fmt UINT16 -smp 1X -pause 15
route -alg msebb -host CEID2 -max 1 -tfm LOG -fmt UINT16 -smp 1X -pause 15
route -alg msebb -host CEID3 -max 1 -tfm LOG -fmt UINT16 -smp 1X -pause 15
route -alg crpm -host CEID11 -max 1 -tfm LOG -fmt UINT16 -smp 1X -pause 15
route -alg pga -host IDEXEC -max 1 -tfm IQ -fmt SCOMPLEX -smp 1X -pause 15 -clean
route -alg pga -host CEID2 -max 1 -tfm IQ -fmt SCOMPLEX -smp 1X -pause 15 -clean
route -alg pga -host CEID3 -max 1 -tfm IQ -fmt SCOMPLEX -smp 1X -pause 15 -clean
route -alg pga -host CEID11 -max 1 -tfm IQ -fmt SCOMPLEX -smp 1X -pause 15 -clean
route -done -host mercury -max 10 -pause 15
tmd -file /atrsys/atr99/config/3algs/jstars.lsd.95.offsets.may00
tmd -file /atrsys/atr99/config/3algs/jstars.lsl.95.offsets.may00
tmd -file /atrsys/atr99/config/3algs/jstars.lsr.95.offsets.may00
tmd -file /atrsys/atr99/config/3algs/jstars.rsd.95.offsets.may00
tmd -file /atrsys/atr99/config/3algs/jstars.rsl.95.offsets.may00
tmd -file /atrsys/atr99/config/3algs/jstars.rsr.95.offsets.may00
```

Appendix C: Interface Requirements

There are two interfaces to the Real-Time ATR System, an input image interface and a result interface. Synthetic aperture radar images are sent to the ATR using the input image interface. The ATR system processes the image and transmits its results to a human machine interface (HMI) or another application using the result interface.

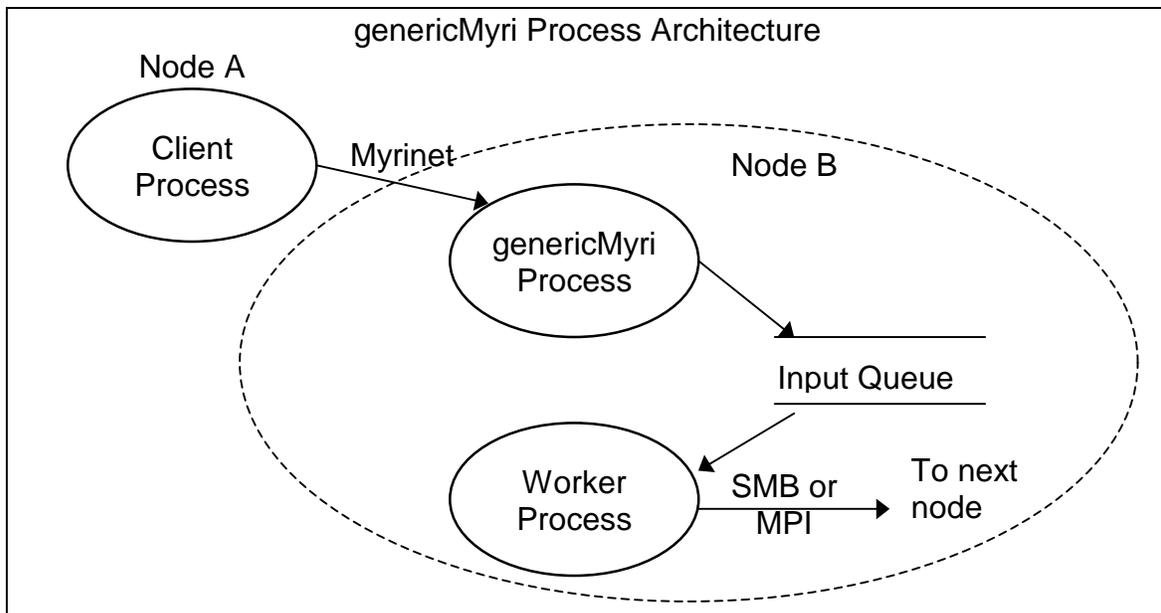


Both interfaces use the Sandia ATR Chip API protocol. The API is a way of managing ATR image data, algorithm cues, and algorithm results within real-time ATR processing systems. It also provides APIs for communicating this data between real-time ATR processing modules. By using a unified API between software modules, the process of connecting modules and building the top-level control structure of an ATR system is simplified. For a tutorial on the ATR Chip API and examples for sending and receiving ATR Chip data, refer to the ATR Chip API User's Manual^{1,2}.

The image and results interfaces can be a Myrinet³ connection or any hardware transport that supports RPC/TCP/IP internet protocols, e.g. 10/100 BaseT Ethernet or FDDI. The advantage of using Myrinet is that the ATR Chip API communication routines use the Myrinet API as the middle layer protocol. This provides a much higher performance (increased bandwidth with lower latencies) interface for sending and receiving ATR Chip data. The disadvantage of its use is that it requires a node in the system, which can accept a Myrinet host adapter interface and is supported by Myricom software drivers.

genericMyri

"genericMyri" is a server process used on the ATR Host to receive image data and algorithm results from external sources. A single instance runs on every node in the system with Myrinet communications. The "genericMyri" process should run at a high priority. This allows the ATR Host to immediately receive incoming data from a client (IFP), thus preventing the client from blocking and hence slowing other client tasks. In addition, communication processing can occur at the same time as computation processing. As messages are received from clients, they are put into a receiving buffer for a worker task, i.e. the Image Former process. When a worker task is ready for a new ATR Chip message, it retrieves it from the receive buffer. After processing the message, the updated ATR Chip information is sent to the next process in the data flow.



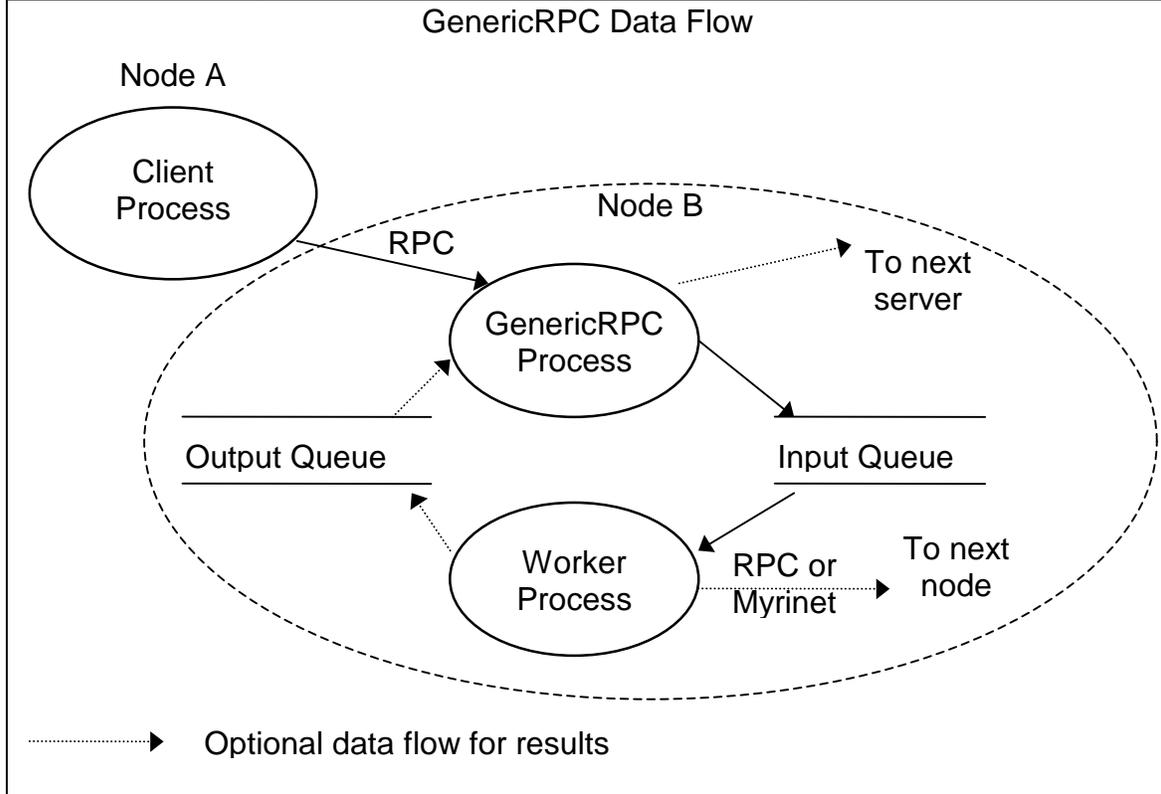
There can only be one "genericMyri" task per node. If there are multiple worker tasks per node, the single "genericMyri" process receives all data and each worker process would have its own Input Queue. The client is able to indicate which buffer to put the data in by specifying a service number. At startup, "genericMyri" is configured with a given number of receive buffers and a service number for each buffer.

There is however a limit to the number of messages the genericMyri task will accept. If the limit is reached, it will no longer accept new messages. In this case the client process will be blocked. This prevents the Host's memory from being used up if it is sent too many messages at one time.

genericRPC (resavRPC)

"GenericRPC" performs the exact same function as "genericMyri" except it receives data sent using the RPC protocol. "GenericRPC" does not have the restriction of only being able to execute a single instance of the process, hence it does not maintain multiple receive buffers. If there are multiple algorithm processes on a node, each process would have its own "genericRPC" process, and associated receive buffer, for receiving data.

"GenericRPC" also maintains a send buffer. This is to support a Send/Receive command. "GenericRPC" places the input data into the receive queue and then blocks on a read of the send queue. The algorithm process reads the receive queue, performs its calculations and then places the results in the send queue. "GenericRPC" then unblocks, reads the send queue and sends the result back to the client. This is useful during algorithm development for debug purposes and also allows non-Myrinet nodes to be integrated into the system.



"resavRPC" is another server process used on the ATR Host to receive ATR results with RPC protocol. It differs from the "genericRPC" server only in respect that it attaches (shares) the Input Queue(s) of another (e. g. "genericMyri") server.

Distribution:

- 1 MS 0844 Drayton Boozer, 15352
- 20 MS 0844 Wallace Bow, 15352
- 1 MS 0844 Brian Bray, 15352
- 1 MS 1110 Douglas Doerfler, 9223
- 2 MS 0980 Richard H. Meyer, 5721
- 1 MS 1110 Neil Pundit, 9223
- 1 MS 9018 Central Technical Files, 8945-1
- 2 MS 0899 Technical Library, 9616
- 1 MS 0612 Review & Approval Desk, 9612
for DOE/OSTI