Superconducting Super Collider Laboratory

CZ/CPS A Communications ZEBRA Implementation Using CPS

L. A. Roberts

May 1991

APPROVED FOR RELEASE OR
PUBLICATION - O.R. PATENT GROUP
BY.....*ES*.....DATE: 4/3/95.

CZ/CPS
A Communications ZEBRA Implementation Using CPS

Lee A. Roberts

Physics Research Division
Superconducting Super Collider Laboratory*
2550 Beckleymeade Avenue
Dallas, Texas 75237

April 1991

*Operated by the Universities Research Association, Inc., for the U.S. Department of Energy under Contract No. DE-AC02-89ER40486

CZ/CPS

A Communications ZEBRA Implementation Using CPS

Lee A. Roberts

Abstract

CZ/CPS is an implementation of the Communications ZEBRA distributed computing environment utilizing the CPS communications protocol. CZ/CPS is intended for parallelization of high energy physics application programs using the CERN Program Library memory and data structure management features. CZ/CPS provides transparent communication of ZEBRA data structures among cooperative processes using standard interfaces for ZEBRA I/O. Examples of usage in a CPS HBOOK4 and GEANT3 application are provided.

DISCLAIMER

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, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents 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 or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

CONTENTS

1.0	INTRODUCTION.....	1
2.0	FEATURES.....	1
2.1	ZEBRA/FZ.....	1
2.1.1	Purpose.....	1
2.1.2	Implementation.....	2
2.1.3	Subprograms.....	4
2.2	HBOOK4/ZEBRA Server.....	6
2.2.1	Purpose.....	6
2.2.2	Implementation.....	6
2.2.3	Subprograms.....	7
2.3	FFREAD	8
2.3.1	Purpose.....	8
2.3.2	Implementation.....	9
2.3.3	Subprograms.....	9
3.0	EXAMPLES.....	11
3.1	Transfer of GEANT3 initialization data structures.....	12
3.1.1	CONTROL process—server.....	12
3.1.2	WORKER process—client.....	13
3.2	Transfer of user-defined ZEBRA data structures.....	14
3.2.1	CONTROL process—server.....	15
3.2.2	WORKER process—client.....	15
3.3	Transfer of HBOOK4 histograms to WORKER processes.....	15
3.3.1	CONTROL process—ZEBRA client.....	16
3.3.2	WORKER process—ZEBRA server.....	17
3.4	Accumulation of HBOOK4 histograms from WORKER processes.....	19
3.4.1	CONTROL process—ZEBRA client.....	19
3.5	Collection of event FZ output	20
3.5.1	OUTPUT process —server.....	20
3.5.2	WORKER process—client.....	22
3.6	Distribution of FFREAD input to WORKER processes.....	23
3.6.1	CONTROL process—server.....	23
3.6.2	WORKER process—client.....	24
4.0	SUMMARY	25
	REFERENCES.....	27

1.0 INTRODUCTION

CPS¹ parallelization of many high-energy physics simulation and analysis codes depends upon the ability to distribute and manage I/O of ZEBRA³ data structures. Users are familiar with the popular HBOOK4⁵ histogramming and data analysis package from the CERN Program Library.² GEANT3⁷ appears to be a popular "standard" for detector description and simulation. Both HBOOK4 and GEANT3 internally use ZEBRA data structures; event-level parallelization of these applications depends upon interprocess communication of ZEBRA data structures.

The CZ/CPS communications package was designed to take full advantage of standard features available in the CERN Program Library codes. Recent development efforts on several of these codes have provided techniques (or "hooks") for interprocess (and interprocessor) communication of data structures based upon a user-specified communications protocol. One example of these communications features is the CSPACK⁶ package for distributed computing, available in a recent release of the CERN Program Library.

The CZ/CPS communication package draws upon the communications ideas and features present in the following packages:

- ZEBRA—FZ Channel I/O
- HBOOK4—CZ Communications Interface
- CSPACK—ZEBRA Server and CZ Communications Interface.

CZ/CPS provides capabilities for communication of ZEBRA data structures, complete with data conversion among unlike processors, using sequential FZ I/O. In addition, HBOOK4 histograms may be manipulated as RZ structures using facilities derived from the handling of global sections on VAX/VMS and the CZ communications software.

In addition, a subset of the programming techniques used for the CZ/CPS implementation may be used to provide distributed FFREAD⁸ format-free input processing for cooperative processes.

2.0 FEATURES

2.1 ZEBRA/FZ

2.1.1 Purpose

ZEBRA/FZ communications provides the basic method for transferring ZEBRA data structures among cooperative processes. ZEBRA/FZ allows transfer of data structures using a sequential medium with proper relocation of pointers and data conversion among incompatible machine architectures.

ZEBRA/FZ communications may be used for distribution of GEANT3 initialization data structures (geometry, materials, particles, etc.), collection of event output data structures into a common FZ output

file, and distribution of FZ event records to analysis processes. Communication of user-defined data structures (magnetic field maps, etc.) may be handled in a straightforward manner using standard FZ I/O interfaces. ZEBRA/FZ also provides the underlying communications for the HBOOK4/ZEBRA Server interface.

2.1.2 Implementation

CZ/CPS communication of ZEBRA/FZ data structures is performed using the FZ Channel I/O medium. Defaults of the FZ Channel medium include Exchange File Format (fixed-length records) and Exchange Data Format (32-bit, big endian, IEEE). These defaults are quite appropriate for the CPS environment, which may consist of a heterogeneous mixture of processors.

Definition of the FZ Channel as a ZEBRA I/O medium is performed using the standard **FZFILE** subroutine interface. Any positive logical unit number may be associated with the FZ Channel; FORTRAN logical unit number restrictions are not imposed. Similarly, selection of the physical record length is up to the user; the standard (default) length should be adequate. CZ/CPS usage should apply the following options to FZ Channel definitions:

- Option 'C' selects medium "Channel."
- Option 'I' selects "Input" direction. Option 'O' selects "Output" direction.
- Option 'S' selects "Separate" data structures.
- Option 'U' selects "Unpacked" data structures.

Option 'S' guarantees that each ZEBRA data structure will be started in a separate physical record. (Earlier ZEBRA releases included option 'S' as a default for media Memory and Channel.) Option 'U' disables the "packing" operation in the exchange format I/O process, leaving it instead under the user's control. On 32-bit computers, "packing" performs byte-swapping, if necessary, to insure that data is in exchange (big endian) format. The CPS communications protocol, which guarantees that 32-bit integer variables are transferred correctly, performs this "packing" operation automatically. The CZ/CPS subroutines take advantage of this feature to minimize the amount of byte-swapping required in ZEBRA data communications.

The FZ Channel medium depends upon an application-specific channel handler subroutine for input/output of data. This channel handler subroutine receives output records from **FZOUT** and provides input records to **FZIN**. Management of the "channel" is handled by this subroutine, which reads or writes FZ physical records on the "channel." In the case of CZ/CPS, two application-specific channel handler subroutines are provided. One of these subroutines (**CZBUFR**) uses a CPS memory block as its "channel," reading or writing information into this local, but globally available, I/O buffer. The second of these subroutines (**CZCPS**) uses CPS subroutines **ACP_GET** or **ACP_SEND** to read or write information on this globally available I/O buffer.

Parameters describing the CPS-specific aspects of the application must be set via subroutine **CZCPSI** for access by the channel management subroutines. This information includes the following:

- **CPSBUF**—CPS block number of globally available I/O buffer.
- **BUFLN**—length (in bytes) of CPS I/O buffer.
- **OFFSET**—offset (in bytes) into CPS I/O buffer.
- **SETPRO**—CPS process number of communicating process.
- **REMSUB**—CPS remote subroutine number for ZEBRA Server.

Not all of these parameters are required in each situation. Table 1 presents an overview of the parameter usage. Current values of these parameters may be retrieved via subroutine **CZCPSR**. (Values are stored in a FORTRAN common block which need not be visible to the user.) The current implementation of CZ/CPS does not provide parameter storage for multiple buffers; it is assumed that a common buffer is used with consistent or reset parameters. Note: It is the user's responsibility to reset the **OFFSET** parameter after each series of FZ I/O transfers.

Table 1. Overview of parameter usage for CPS-specific aspects of the CZ/CPS interface. User must provide values for entries labeled "user;" "system" values are provided automatically. Some parameters are not always applicable or are client-mode only.

Parameter	ZEBRA/FZ	ZEBRA/FZ	HBOOK4/ZEBRA Server	HBOOK4/ZEBRA Server
	CZBUFR	CZCPS	CZBUFR	CZCPS
CPSBUF	n/a	user	n/a	user
BUFLN	user	user	user	user
OFFSET	user	user	system	system
SETPRO	n/a	user	n/a	system
REHSUB	n/a	n/a	user/client	user/client

Definition of the channel handler subroutine to ZEBRA is performed using the **FZHOOK** subroutine. This subroutine associates an I/O stream with the desired channel manager after the "file" has been initialized with **FZFILE** but before any I/O operations are attempted with **FZIN** or **FZOUT**.

Communication of ZEBRA FZ information using CZ/CPS must be coordinated among server and client processes. Multiple clients may read from a single server without interference. This capability allows a substantial savings in memory usage, since only the server process requires a large CPS memory buffer, rather than duplication in each client. Synchronization points (as provided by **ACP_SYNC**) must be used to coordinate access. Obviously, clients should not begin reading until the server has placed the information into the buffer. Similarly, the server must not overwrite the buffer until all clients have finished reading the information. The examples shown in a later section demonstrate these features.

Complete documentation on usage of the FZ Channel medium and **FZHOOK** may be found in the *ZEBRA Reference Manual*.⁴

2.1.3 Subprograms

CZBUFR

Description: Subroutine **CZBUFR** performs I/O of FZ physical records on a local memory buffer in FORTRAN common block **CCZBUF**. The size of this memory buffer must be set by the user. Declaration of this memory buffer as a CPS I/O block must be performed by the user. Subroutine **CZBUFR** allows multiple ZEBRA data structures to be read or written sequentially on a local memory buffer. Subroutine **CZBUFR** also provides the message-passing interface used by the CPS ZEBRA Server. CZ/CPS subroutine **CZBUFR** is functionally equivalent, but with lower overhead, to subroutine **CZCPS** operating on a local CPS memory buffer.

Interface:

```
SUBROUTINE CZBUFR (IBUF, IOWAY)
  INTEGER IBUF (*), IOWAY (2)
  COMMON /CCZCPS/ ICZOFF, ICZSET, ICZCPS, ICZBUF, ICZRS
  COMMON /CCZBUF/ MCZBUF (1)
```

Usage: Not user callable. Must be declared **EXTERNAL** and defined for FZ I/O via **FZHOOK**. CZ/CPS interface parameters must be set using subroutine **CZCPSI**.

```
EXTERNAL CZBUFR
CALL FZFILE (LUN, CZBLK, 'COSU')
CALL FZHOOK (LUN, CZBUFR, IDUMMY)
```

CZCPS

Description: Subroutine **CZCPS** performs I/O of FZ physical records on a remote CPS memory buffer. The size of this memory buffer must be set by the user. Declaration of this memory buffer as a CPS I/O block should be performed by the user. Subroutine **CZCPS** allows multiple ZEBRA data structures to be read or written sequentially on a remote CPS memory buffer using **ACP_GET** or **ACP_SEND**. Subroutine **CZCPS** also provides the message-passing interface used by the CPS ZEBRA Server. CZ/CPS subroutine **CZCPS** is functionally equivalent to CSPACK subroutine **CZTCP**, but uses CPS as a communications protocol rather than TCP/IP.

Interface:

```
SUBROUTINE CZCPS (IBUF, IOWAY)
  INTEGER IBUF (*), IOWAY (2)
  COMMON /CCZCPS/ ICZOFF, ICZSET, ICZCPS, ICZBUF, ICZRS
```

Usage: Not user callable. Must be declared **EXTERNAL** and defined for FZ I/O via **FZHOOK**. CZ/CPS interface parameters must be set using subroutine **CZCPSI**.

```
EXTERNAL CZCPS
CALL FZFILE (LUN, CZBLK, 'CISU')
CALL FZHOOK (LUN, CZCPS, IDUMMY)
```

CZCPSI

Description: Initialization interface for variables in FORTRAN common block **CCZCPS**.

Interface:

```
SUBROUTINE CZCPSI (NAME, VALUE)
CHARACTER*6 NAME
INTEGER VALUE
COMMON /CCZCPS/ ICZOFF, ICZSET, ICZCPS, ICZBUF, ICZRS
```

Usage: User and CZ/CPS internal. **NAME** may assume the following values:

- **CPSBUF**—CPS block number of globally available I/O buffer.
- **BUFLEN**—length (in bytes) of CPS I/O buffer.
- **OFFSET**—offset (in bytes) into CPS I/O buffer.
- **SETPRO**—CPS process number of communicating process.
- **REMSUB**—CPS remote subroutine number for ZEBRA Server.

Not all of these parameters are required in each situation. Table 1 presents an overview of the parameter usage.

CZCPSR

Description: Retrieval interface for variables in FORTRAN common block **CCZCPS**.

Interface:

```
SUBROUTINE CZCPSR (NAME, VALUE)
CHARACTER*6 NAME
INTEGER VALUE
COMMON /CCZCPS/ ICZOFF, ICZSET, ICZCPS, ICZBUF, ICZRS
```

Usage: User and CZ/CPS internal. **NAME** may assume the following values:

- **CPSBUF**—CPS block number of globally available I/O buffer.
- **BUFLEN**—length (in bytes) of CPS I/O buffer.

- **OFFSET**—offset (in bytes) into CPS I/O buffer.
- **SETPRO**—CPS process number of communicating process.
- **REMSUB**—CPS remote subroutine number for ZEBRA Server.

Not all of these parameters are required in each situation. Table 1 presents an overview of the parameter usage.

2.2 HBOOK4/ZEBRA Server

2.2.1 Purpose

HBOOK4/ZEBRA Server communications provide a convenient method for transfer and manipulation of HBOOK4 histograms in cooperative processes. An interface similar to VAX/VMS global sections and RZ direct-access files is provided for the user. In a typical CPS job, the user may declare (book) histograms in an initialization process, use **HROUT** to write these histograms to each worker process, analyze data, and finally use **HRIN** to read and sum the histograms from each of the worker processes.

2.2.2 Implementation

HBOOK4/ZEBRA Server communications depend upon the ZEBRA/FZ communications described in the previous section. Two FZ Channel media are used: logical unit 998 for output and logical unit 999 for input. At the present release of HBOOK4, these values are coded directly into the HBOOK4/CZ interface. Both FZ Channel media must be properly defined before HBOOK4/ZEBRA Server I/O is attempted. However, unlike the ZEBRA/FZ communications described in the previous section, the user need not be concerned with resetting the I/O buffer offset or the communicating process number; these are reset automatically. Synchronization of communicating processes is handled automatically via call-and-wait usage of remote subroutine calls.

The CPS ZEBRA Server, **ZSCPS**, must be accessed via remote subroutine call. The user must define **ZSCPS** as a remote entry point to the CPS software using **ACP_DECLARE_SUBROUTINE** and set the remote subroutine number for the CZ/CPS software using **CZCPSI**. All ZEBRA Server requests are handled in a method quite similar to the standard ZEBRA Server available in **CSPACK**.

HBOOK4/ZEBRA Server operations return the **IQUEST** vector of the remote process. Therefore, HBOOK4/ZEBRA Server operations must occur between pairs of processes—a single client and a single server—otherwise confusion will result from multiple servers attempting to write into the single I/O buffer on the client process.

Supported HBOOK4 operations include the following:

- **HRFILE**—initiate connection to specified CPS process.
- **HREND**—terminate connection to specified CPS process.

- **HROUT**—write histograms to current CPS process. Users may select the following options:
 - Write a single histogram to the remote process.
 - Write all histograms in the current directory to the remote process.
 - Write a directory “tree” of histograms to the remote process.
- **HRIN**—read histograms from current CPS process. Users may select the following options:
 - Read a single histogram from the remote process.
 - Read all histograms from the remote current directory.
 - Add histograms if offset is 99999.
- **HMDIR**—make a subdirectory of the remote current directory.
- **HCDIR**—change the remote current directory.
- **HRDIR**—return a list of subdirectories of the remote current directory.

2.2.3 Subprograms

ZSCPS

Description: Subroutine **ZSCPS** is a ZEBRA Server for CPS cooperative processes which supports remote HBOOK4 operations on histograms in its local memory. **ZSCPS** must be accessed as a remote subroutine call. Supported operations include remote input and output of histograms and remote directory manipulation.

Auxiliary entry points **ZSCPSI**, **ZSCPSO**, **ZSCPSM**, **ZSCPSC** and **ZSCPSR** perform tasks for the control subroutine.

Interface:

```
SUBROUTINE ZSCPS (IMAIL, IPROC, ISOURC)
INTEGER IMAIL(20), IPROC, ISOURC
COMMON /CCZCPS/ ICZOFF, ICZSET, ICZCPS, ICZBUF, ICZRS
```

Usage: Not user callable. Must be declared **EXTERNAL** and declared a CPS remote subroutine.

```
EXTERNAL ZSCPS
CALL ACP_DECLARE_SUBROUTINE (ZSCPS, RZSCPS, 3, 80, 4, 4)
```

HRFILE

Description: Customization of CZ-specific code present in standard HRFILE to handle CZ/CPS communications. Adapt code for VAX/VMS global sections to handle CZ/CPS communicating processes via ZEBRA Server.

Interface:

```
SUBROUTINE HRFILE (LUN, CHDIR, CHOPT)
INTEGER LUN
CHARACTER* (*) CHDIR, CHOPT
```

Usage: User. **LUN** must specify the CPS process number of the server process. Use option 'G'.

HROUT

Description: Customization of CZ-specific code present in standard **HROUT** to handle CZ/CPS communications. Supports standard CZ operations.

Interface:

```
SUBROUTINE HROUT (IDD, ICYCLE, CHOPT)
INTEGER IDD, ICYCLE
CHARACTER* (*) CHOPT
```

Usage: User. Use options 'TN' when writing a directory "tree" of histograms to a remote process.

HMDIR, HRDIR, HRZCD & HRZOUT

Description: Customization of CZ-specific code present in standard subroutines to handle CZ/CPS communications.

Interface: No change from standard implementation.

Usage: No change from standard implementation.

2.3 FFREAD

2.3.1 Purpose

FFREAD is widely used in applications referencing the CERN Program Library as a free-format input processing interface for application data cards. GEANT3 is among these applications, where a standard set of "system" FFREAD data cards and optionally user-defined FFREAD data cards are used to control initialization. In a typical application, a variety of common blocks are affected by the combination of "system" and user FFREAD cards. When attempting CPS distribution of such initialization, one faces the task of broadcasting the effects of these many data cards—typically covering many common blocks—and certainly varying from user to user. Or, one may choose to broadcast the FFREAD data cards—and let each cooperative process perform the initialization itself.

A consistent interface for all FFREAD data cards—"system" or user—in CPS GEANT3 applications may be achieved via the second approach. Broadcast of FFREAD data cards to each cooperative process—with each performing its own initialization should be acceptable as long as initialization times are reasonably short.

2.3.2 Implementation

Communication of FFREAD data cards from an initialization process to a set of cooperative worker processes can be handled quite easily. FFREAD data cards intended for the worker processes may be loaded into a CPS memory block as fixed-length records (of length LINPUT as set within FFREAD) for global access from the worker processes. Access from the worker processes may be handled by an alternate input routine, **FFCPS**, which "reads" the data cards in the CPS memory block using **ACP_GET**. Definition of this alternate input routine is handled by the standard FFREAD subroutines **FFSET** and **FFRSET**.

Parameters describing the CPS-specific aspects of the application must be set via subroutine **FFCPSI** for access by the alternate input routine. This information includes the following:

- **CPSBUF**—CPS block number of globally-available I/O buffer
- **BUFLN**—length (in bytes) of CPS I/O buffer
- **OFFSET**—offset (in bytes) into CPS I/O buffer
- **SETPRO**—CPS process number of communicating process.

Current values of these parameters may be retrieved using subroutine **FFCPSR**. (Values are stored in a FORTRAN common block which need not be visible to the user.) The current implementation of CZ/CPS does not provide parameter storage for multiple buffers; it is assumed that a common buffer is used with consistent or reset parameters.

Communication of FFREAD data cards using CZ/CPS must be coordinated among server and client processes. Appropriate use of synchronization points (as provided by **ACP_SYNC**) can coordinate access to the memory buffer.

2.3.3 Subprograms

FFCPS

Description: Subroutine **FFCPS** performs input of FFREAD data cards from a remote memory buffer. The size of this memory buffer must be set by the user. Declaration of this memory buffer as a CPS I/O block should be performed by the user. Subroutine **FFCPS** uses **ACP_GET** to access each FFREAD data card. Each data card is assumed to be a fixed-length record of length LINPUT as set within FFREAD.

Interface:

```
SUBROUTINE FFCPS (IBFSIZ)
  INTEGER IBFSIZ
```

Usage: Not user callable. Must be declared **EXTERNAL** and defined as an alternate **FFREAD** input routine via **FFSET** and **FFRSET**. CPS interface parameters must be set using subroutine **FFCPSI**.

```
EXTERNAL FFCPS
CALL FFSET ('LINP', -FFLUN)
CALL FFRSET (FFLUN, FFCPS)
```

FFCPSI

Description: Initialization interface for variables in FORTRAN common block **CFFCPS**.

Interface:

```
SUBROUTINE FFCPSI (NAME, VALUE)
  CHARACTER*6 NAME
  INTEGER VALUE
  COMMON /CFFCPS/ IFFOFF, IFFSET, IFFCPS, IFFBUF
```

Usage: User. **NAME** may assume the following values:

- **CPSBUF**—CPS block number of globally-available I/O buffer.
- **BUFLN**—length (in bytes) of CPS I/O buffer.
- **OFFSET**—offset (in bytes) into CPS I/O buffer.
- **SETPRO**—CPS process number of communicating process.

FFCPSR

Description: Retrieval interface for variables in FORTRAN common block **CFFCPS**.

Interface:

```
SUBROUTINE FFCPSR (NAME, VALUE)
  CHARACTER*6 NAME
  INTEGER VALUE
  COMMON /CFFCPS/ IFFOFF, IFFSET, IFFCPS, IFFBUF
```

Usage: User. **NAME** may assume the following values:

- **CPSBUF**—CPS block number of globally available I/O buffer.
- **BUFLN**—length (in bytes) of CPS I/O buffer.

- **OFFSET**—offset (in bytes) into CPS I/O buffer.
- **SETPRO**—CPS process number of communicating process.

3.0 EXAMPLES

Several application examples of CZ/CPS are now presented. These examples have been derived from a CPS event-level parallelization of the Bottom Collider Detector (BCD) GEANT3 simulation.⁹

In its present implementation, the CPS parallelization of the BCD GEANT3 simulation takes the following form. Each process performs its own basic initialization of ZEBRA, GEANT3 (**GZEBRA**, **GINIT**) and HBOOK4 (**HLIMIT**), if required.

- **CONTROL** process.
 - Reads and distributes FFREAD data cards for **WORKER** processes.
 - Generates and distributes GEANT3 initialization data structures (geometry, materials, particles, etc.) to **WORKER** processes.
 - Reads (via FZ) and distributes field maps of two dipole magnets to **WORKER** processes.
 - Creates and distributes HBOOK4 analysis histograms to **WORKER** processes.
 - Starts event generation servers.
 - Starts FZ output collector.
 - Controls GEANT3 event loop, distributing run number, event number and random generator seeds via remote subroutine calls to **WORKER** processes.
 - Accumulates HBOOK4 histograms from **WORKER** processes at end of run.
 - Closes FZ output collector.
- **EVTGEN** processes.
 - ISAJET¹⁰ or PYTHIA¹¹ event generation servers.
 - Interface via remote subroutine call.
 - Provide events for **WORKER** processes.
 - Number of event generation servers may be adjusted to meet computational demand or physics needs.
 - Could be adapted into disk/tape input event servers.
- **OUTPUT** processes.
 - FZ output collector.

- Interface via ZEBRA/FZ and remote subroutine call.
- Number of collectors may be adjusted to meet output load.
- **WORKER** processes.
 - BCD GEANT3 event generation server.
 - CPS ZEBRA Server.
 - Interface via remote subroutine call.
 - Request services from **EVTGEN** and **OUTPUT** servers.

3.1 Transfer of GEANT3 initialization data structures

This example demonstrates the transfer of GEANT3 initialization data structures from an initialization (or control) process to a set of worker processes. Code fragments for the initialization and worker processes are shown in the following subsections. We assume that the initialization process has the GEANT initialization data structures available in its local ZEBRA store.

The code fragments display the following steps:

- Perform CPS initialization in **CONTROL** and **WORKER** processes. Write GEANT3 initialization data structures into memory buffer in **CONTROL** process.
- Synchronization point. **WORKER** processes may now read information.
- **WORKER** processes read GEANT3 initialization data structures from **CONTROL** process.
- Synchronization point. **WORKER** processes have read information.
- All processes advance to next task.

3.1.1 CONTROL process—server

C Define CPS block number for FZ channel transfers.

```
INTEGER CPSFZ
PARAMETER (CPSFZ = 1)
```

C Define FZ channel buffer size (words). Buffer is local.

C Adjust buffer size as necessary for individual applications.

```
INTEGER CZBUF
PARAMETER (CZBUF = 262144)
```

C Declare CPS memory buffer for FZ Channel I/O.

```
INTEGER MCZBUF
COMMON /CCZBUF/ MCZBUF (CZBUF)
```

```

C   Define block size (words) for FZ channel data transfers.
C   900 words is ZEBRA default; must be multiple of 90 words.

      INTEGER CZBLK
      PARAMETER (CZBLK = 900)

C   Declare channel handler subroutine. CPS block is local.
C   Use channel handler for local CPS memory block.

      EXTERNAL CZBUFR

C   Initialize CPS software.

      CALL ACP_INIT

C   Define CPS memory block for FZ channel communications via CPS.
C   Store buffer length (bytes) for channel handler.

      CALL ACP_DECLARE_BLOCK(MCZBUF,4*CZBUF,CPSFZ)
      CALL CZCPSI('BUFLN',4*CZBUF)

C   Transfer GEANT data structures to local memory buffer.

      CALL CZCPSI('OFFSET', 0)
      CALL GOPEN(LUNFZ, 'COSU',CZBLK,IERROR)
      CALL FZHOOK(LUNFZ,CZBUFR,IDUMMY)
      CALL GSAVE(LUNFZ,'INIT', -1,0,IERROR)
      CALL GCLOSE(LUNFZ,IERROR)

C   Synchronize CONTROL and WORKER processes after data structure
C   transfer to local memory buffer.

      CALL ACP_SYNC(ACP_SET_OF_PROCESSES(2,CONTROL,WORKER),1)

C   Synchronize CONTROL and WORKER processes after WORKER processes have
C   completed transfers from local memory buffer.

      CALL ACP_SYNC(ACP_SET_OF_PROCESSES(2,CONTROL,WORKER),2)

3.1.2 WORKER process—client

C   Define CPS block number for FZ channel transfers.

      INTEGER CPSFZ
      PARAMETER (CPSFZ = 1 )

C   Define FZ channel buffer size (words). Buffer is remote.
C   Adjust buffer size as necessary for individual applications.

      INTEGER CZBUF
      PARAMETER ( CZBUF = 262144 )

```

```

C   Define block size (words) for FZ channel data transfers.
C   900 words is ZEBRA default; must be multiple of 90 words.

      INTEGER CZBLK
      PARAMETER ( CZBLK = 900 )

C   Declare channel handler subroutine. CPS block is remote.
C   Use channel handler with ACP_GET and ACP_SEND for remote access.

      EXTERNAL CZCPS

C   Initialize CPS software.

      CALL ACP_INIT

C   Define CPS block information for FZ channel communications via CPS.
C   CPS block is remote. Store CPS block number, buffer length (bytes) and
C   CPS process number of CONTROL process.

      CALL CZCPSI('CPSBUF',CPSFZ)
      CALL CZCPSI('BUFLN',4*CZBUF)
      CALL ACP_JOB_INFO(1,CONTROL,IPROC1,IPROCN)
      CALL CZCPSI('SETPRO',IPROC1)

C   Synchronize CONTROL and WORKER processes to allow data structures to be
C   written into memory buffer on CONTROL process.

      CALL ACP_SYNC(ACP_SET_OF_PROCESSES(2,CONTROL,WORKER),1)

C   Read GEANT Initialization data structures from FZ channel.

      CALL CZCPSI('OFFSET',0)
      CALL GOPEN(LUNFZ,'CISU',CZBLK,IERROR)
      CALL FZHOOK(LUNFZ,CZCPS,IDUMMY)
      CALL GGET(LUNFZ,'INIT',-1,IDENT,IERROR)
      CALL GCLOSE(LUNFZ,IERROR)

C   Synchronize processes when transfers have finished.

      CALL ACP_SYNC(ACP_SET_OF_PROCESSES(2,CONTROL,WORKER),2)

```

3.2 Transfer of user-defined ZEBRA data structures

This example demonstrates the transfer of user-defined ZEBRA data structures from an initialization (or control) process to a set of worker processes. Code fragments for the initialization and worker processes are shown in the following subsections. We assume that the initialization process has the user's data structures available in its local ZEBRA store. We also assume that the code fragments of the previous example have been executed prior to the present code fragments.

In this example, the user has two data structures, the magnetic field maps of two dipole magnets, at JBFLD (1) and JBFLD (2) in ZEBRA division IXCONS. This example demonstrates the transfer of multiple data structures within a single CZ/CPS "file." We use the CPS memory buffer as defined in the previous example—only the offset into the memory buffer need be reset. Synchronization points are used to coordinate the data transfer, as in the previous example.

3.2.1 CONTROL process—server

C Transfer magnetic field maps to WORKER processes, if necessary.

```

IF (IFIELD.EQ.1) THEN
  CALL CZCPSI('OFFSET',0)
  CALL FZFILE(LUNFZ,CZBLK,'COSU')
  CALL FZHOOK(LUNFZ,CZBUFR,IDUMMY)
  CALL FZOUT(LUNFZ,IXCONS,JBFLD(1),0,' ',2,0,0)
  CALL FZOUT(LUNFZ,IXCONS,JBFLD(2),0,' ',2,0,0)
  CALL FZENDO(LUNFZ,'T')
  CALL ACP_SYNC(ACP_SET_OF_PROCESSES(2,CONTROL,WORKER),3)
  CALL ACP_SYNC(ACP_SET_OF_PROCESSES(2,CONTROL,WORKER),4)
ENDIF

```

3.2.2 WORKER process—client

C Transfer magnetic field maps to WORKER process, if necessary.

```

IF (IFIELD.EQ.1) THEN
  CALL ACP_SYNC(ACP_SET_OF_PROCESSES(2,CONTROL,WORKER),3)
  CALL CZCPSI('OFFSET',0)
  CALL FZFILE(LUNFZ,CZBLK,'CISU')
  CALL FZHOOK(LUNFZ,CZCPS,IDUMMY)
  CALL FZIN(LUNFZ,IXCONS,JBFLD(1),1,' ',0,0)
  CALL FZIN(LUNFZ,IXCONS,JBFLD(2),1,' ',0,0)
  CALL FZENDI(LUNFZ,'T')
  CALL ACP_SYNC(ACP_SET_OF_PROCESSES(2,CONTROL,WORKER),4)
ENDIF

```

3.3 Transfer of HBOOK4 histograms to WORKER processes

This example demonstrates the transfer of HBOOK4 histograms from an initialization (or control) process to a set of worker processes. Code fragments for the initialization and worker processes are shown in the following subsections. We assume that the user's histograms are available in the local ZEBRA store of the initialization process.

Most of the CZ/CPS initialization procedure shown in this example is identical to that of the first example. Note that two FZ Channel media are defined with logical units 998 (output) and 999 (input).

Both may share a common I/O buffer, since no simultaneous bidirectional data transfers occur. Management of the buffer offset is internal to the HBOOK4/ZEBRA Server, unlike the case of FZ transfers shown in the previous examples.

3.3.1 CONTROL process—ZEBRA client

C IADCPS in common CZSOCK must be set for HBOOK4/ZEBRA Server.

```
COMMON /CZSOCK/ LUNCZ,IADCPS
```

C Define parameters for CPS remote subroutines.

```
INTEGER RZSCPS  
PARAMETER ( RZSCPS = 1 )
```

C Define FZ channel transfer "logical units".

C Use values consistent with HBOOK4/CZ interface.

```
INTEGER CZLUNI  
PARAMETER ( CZLUNI = 999 )  
INTEGER CZLUNO  
PARAMETER ( CZLUNO = 998 )
```

C Define CPS block number for FZ channel transfers.

```
INTEGER CPSFZ  
PARAMETER ( CPSFZ = 1 )
```

C Define FZ channel buffer size (words). Buffer is local.

C Adjust buffer size as necessary for individual applications.

```
INTEGER CZBUF  
PARAMETER ( CZBUF = 262144 )
```

C Declare CPS memory buffer for FZ Channel I/O.

```
INTEGER MCZBUF  
COMMON /CCZBUF/ MCZBUF(CZBUF)
```

C Define block size (words) for FZ channel data transfers.

C 900 words is ZEBRA default; must be multiple of 90 words.

```
INTEGER CZBLK  
PARAMETER ( CZBLK = 900 )
```

C Declare channel handler subroutine. CPS block is local.

C Use channel handler for local CPS memory block.

```
EXTERNAL CZBUFR
```

```

C   Declare character variable for RZ top directory name.

      CHARACTER*4 CHTOP

C   Initialize CPS software.

C   Declare CPS block for FZ channel communications via CPS.

C   Store buffer length (bytes) for channel handler.

      CALL ACP_DECLARE_BLOCK(MCZBUF,4*CZBUF,CPSFZ)
      CALL CZCPSI('BUFLN',4*CZBUF)

C   Initialize FZ Channel Input and Output media for HBOOK4/ZEBRA
C   Server communications.

      CALL FZFILE(CZLUNO,CZBLK,'COSU')
      CALL FZFILE(CZLUNI,CZBLK,'CISU')
      CALL FZHOOK(CZLUNO,CZBUFR,IDUMMY)
      CALL FZHOOK(CZLUNI,CZBUFR,IDUMMY)

C   Define channel handler address for CZ communications interface.

      IADCPS=JUMPAD(CZBUFR)
      CALL JUMPST(IADCPS)

C   Define CPS remote subroutine number of ZEBRA Server in WORKER process.

      CALL CZCPSI('REMSUB',RZSCPS)

C   Loop over all WORKER processes.
C   Open each WORKER process with option 'G'.
C   Write all histograms in current directory to each WORKER.
C   Close each WORKER process when finished.

      CALL ACP_JOB_INFO(1,WORKER,ISTART,ICOUNT)
      DO 100 IPROC = ISTART, ISTART + ICOUNT-1
        WRITE(CHTOP,'(I4.4)') IPROC
        CALL HRFILE(IPROC,CHTOP,'G')
        CALL HROUT(0,ICYCLE,' ')
        CALL HREND(CHTOP)
      100 CONTINUE

```

3.3.2 WORKER process—ZEBRA server

```

C   IADCPS in common CZSOCK must be set for HBOOK4/ZEBRA Server.

      COMMON /CZSOCK/ LUNCZ,IADCPS

C   Define parameters for CPS remote subroutines.

```



```

INTEGER RZSCPS
PARAMETER ( RZSCPS = 1 )

C   Define FZ channel transfer "logical units"
C   Use values consistent with HBOOK4/CZ interface.

INTEGER CZLUNI
PARAMETER ( CZLUNI = 999 )
INTEGER CZLUNO
PARAMETER ( CZLUNO = 998 )

C   Define CPS block number for FZ channel transfers.

INTEGER CPSFZ
PARAMETER ( CPSFZ = 1 )

C   Define FZ channel buffer size (words). Buffer is remote.
C   Adjust buffer size as necessary for individual applications.

INTEGER CZBUF
PARAMETER ( CZBUF = 262144 )

C   Define block size (words) for FZ channel data transfers.
C   900 words is ZEBRA default; must be multiple of 90 words.

INTEGER CZBLK
PARAMETER ( CZBLK = 900 )

C   Declare channel handler subroutine. CPS block is remote.
C   Use channel handler with ACP_GET and ACP_SEND for remote access.

EXTERNAL CZCPS

C   External declaration for ZEBRA Server.

EXTERNAL ZSCPS

C   Initialize CPS software.
C   Declare ZEBRA Server as a remote subroutine.

CALL ACP_INIT
CALL ACP_DECLARE_SUBROUTINE(ZSCPS,RZSCPS,3,80,4,4)

C   Define CPS block information for FZ channel communications via CPS.
C   CPS block is remote. Store CPS block number and buffer length (bytes).

CALL CZCPSI('CPSBUF',CPSFZ)
CALL CZCPSI('BUFLen',4*CZBUF),

C   Initialize FZ Channel Input and Output media for HBOOK4/ZEBRA
C   Server communications.

```

```

CALL FZFILE(CZLUNO,CZBLK,'COSU')
CALL FZFILE(CZLUNI,CZBLK,'CISU')
CALL FZHOOK(CZLUNO,CZCPS,IDUMMY)
CALL FZHOOK(CZLUNI,CZCPS,IDUMMY)

```

C Define channel handler address for CZ communications interface.

```

IADCPS = JUMPAD(CZCPS)
CALL JUMPST(IADCPS)

```

C Service remote subroutine calls.

```

CALL ACP_SERVICE_CALLS

```

3.4 Accumulation of HBOOK4 histograms from WORKER processes

This example shows the accumulation of HBOOK4 histograms from the set of **WORKER** processes at job termination. We assume that the conditions of the previous example hold—the **CONTROL** process continues to have the initial set of (empty) histograms that were written to each of the **WORKER** processes for filling. We also assume that the finished **WORKER** processes have been queued on the **DONE_QUEUE** with an **ACP\$END_OF_QUEUE** marker.

Only one cycle of each histogram is supported under CZ/CPS, as the remote “file” is actually a ZEBRA memory store. The special offset of 99999 causes the input histograms to be added to those in memory.

No code segment is shown for the **WORKER** process. The remote subroutine server of the previous example is adequate.

3.4.1 CONTROL process—ZEBRA client

C Collect histograms from WORKER processes.

```

C For each WORKER,
C   remove from DONE_QUEUE,
C   open ZEBRA Server connection,
C   input and add all histograms in top directory,
C   close ZEBRA Server connection.

```

100 CONTINUE

```

CALL ACP_DEQUEUE_PROCESS(IPROC,DONE_QUEUE)
IF (IPROC.EQ.ACP$END_OF_QUEUE) GOTO 200
WRITE(CHTOP,'(I4.4)') IPROC
CALL HRFILE(IPROC,CHTOP,'G')
CALL HRIN(0,1,99999)
CALL HREND(CHTOP)
GOTO 100

```

200 CONTINUE

3.5 Collection of event FZ output

This example demonstrates the collection of FZ event output from GEANT3 by an **OUTPUT** server. When each **WORKER** process enters subroutine **GUOUT**, an **OUTPUT** server is dequeued. The FZ event output is transferred to the **OUTPUT** server's local memory buffer for output to disk or tape. The data transfer to disk or tape is done asynchronously as the **WORKER** process proceeds to its next event. Since all data transfer is performed using binary exchange format, output may be collected from a heterogeneous set of **WORKER** processes to form an FZ binary exchange format file. Note: This **OUTPUT** server is designed for a 32-bit, big endian machine. Additional byte-swapping is required before writing data to disk or tape when running on a 32-bit, little endian machine.

3.5.1 OUTPUT process—server

```
INCLUDE 'acp_user.inc'
```

C Define parameters for CPS process queues.

```
INTEGER QOUTPUT
PARAMETER ( QOUTPUT = 1 )
```

C Define parameters for CPS remote subroutines.

```
INTEGER RCREAT, ROUTBL, RFINIS
PARAMETER ( RCREAT = 1 )
PARAMETER ( ROUTBL = 2 )
PARAMETER ( RFINIS = 3 )
```

C Define CPS block number for FZ channel transfers.

```
INTEGER CPSFZ
PARAMETER ( CPSFZ = 1 )
```

C Define FZ channel buffer size (words). Buffer is local.

C Adjust buffer size as necessary for individual applications.

```
INTEGER CZBUF
PARAMETER ( CZBUF = 262144 )
```

C Declare CPS memory buffer for FZ Channel I/O.

```
INTEGER MCZBUF
COMMON /CCZBUF/ MCZBUF(CZBUF)
```

C Declare CPS remote subroutines.

```
EXTERNAL CREATE
EXTERNAL OUTBLK
EXTERNAL FINISH
```

C Initialize CPS software.

```
CALL ACP_INIT
CALL ACP_DECLARE_BLOCK(MCZBUF, 4*CZBUF, CPSFZ)
CALL ACP_DECLARE_SUBROUTINE(CREATE, RCREAT, 5, 80, 4, 4, 4, 4)
CALL ACP_DECLARE_SUBROUTINE(OUTBLK, ROUTBL, 1, 4)
CALL ACP_DECLARE_SUBROUTINE(FINISH, RFINIS, 0)
CALL ACP_DECLARE_QUEUE(QOUTPUT, 1, 4)

CALL ACP_SYNC(ACP$ALL_PROCESSES, 0)
```

C Service CPS remote subroutine calls.

```
CALL ACP_SERVICE_CALLS

END

SUBROUTINE CREATE(NAME, MEDIUP, LRECP, NBUFP, SOURCE)
```

C Open the FZ output file.

```
INTEGER NAME(*), INAME(20), MEDIUP, LRECP, NBUFP, SOURCE, TARGET
CHARACTER*80 CHNAME

INCLUDE 'acp_user.inc'

INTEGER LUNPTR, MEDIUM, LRECL, NBUF
COMMON / FILE / LUNPTR, MEDIUM, LRECL, NBUF

MEDIUM = MEDIUP
LRECL = LRECP
NBUF = NBUFP
CALL ACP_CLASS_INFO(TARGET, IPROC1, IPROC2)
CALL ACP_CONVERT(NAME, INAME, 80, ACP$CHARACTER, SOURCE, TARGET)
CALL UHTOC(INAME, 4, CHNAME, 80)
LENGTH = LENOC(CHNAME)

CALL CFOPEN(LUNPTR, MEDIUM, LRECL, 'w', NBUF, CHNAME(1:LENGTH), ISTAT)

END

SUBROUTINE OUTBLK(NPHR)
```

C Write NPHR physical records of length LRECL to the FZ file.

```
INTEGER NPHR

INTEGER MCZBUF
COMMON /CCZBUF/ MCZBUF(1)
```

```

      INTEGER LUNPTR, MEDIUM, LRECL, NBUF
      COMMON / FILE / LUNPTR, MEDIUM, LRECL, NBUF

      DO 100 I=0,NPHR-1
        CALL CFPUT(LUNPTR,MEDIUM,LRECL,MCZBUF(1+I*LRECL),ISTAT)
100 CONTINUE

```

END

SUBROUTINE FINISH

```

      INTEGER LUNPTR, MEDIUM, LRECL, NBUF
      COMMON / FILE / LUNPTR, MEDIUM, LRECL, NBUF

```

C Close the FZ output file.

```

      CALL CFCLOS(LUNPTR,MEDIUM)

```

END

3.5.2 WORKER process—client

* We assume that the CONTROL process has opened the FZ output file using
 * the following code fragment.

```

*     CHNAME = 'geant.fz'
*     CALL UCTOH(CHNAME, INAME, 4, 80)
*     IMED = 0
*     IBLOCK = CZBLK
*     INBUF = 0
*     ICLASS = CONTROL
*     CALL ACP_CALL(IPROC, ACP$WAIT, RCREAT,
*     # INAME, IMED, IBLOCK, INBUF, ICLASS)

```

* We also assume that the WORKER process has previously opened the FZ
 * connection for event output using the following code fragment.

```

*     CALL CZCPSI ( 'CPSBUF', CPSFZ)
*     CALL CZCPSI('BUFLN' , 4*CZBUF)
*     CALL GOPEN(LUNITS(2), 'COSU' , CZBLK, IERROR)
*     CALL FZHOOK(LUNITS(2) , CZCPS, IDUMMY)

```

* The following code fragment from SUBROUTINE GUOUT of WORKER process
 * demonstrates the use of an OUTPUT server to collect GEANT FZ output.

C Attach an output process

```

      CALL ACP_DEQUEUE_PROCESS(IPROC, QOUTPUT, NPHR)

```

C Transfer event output to output process

```

CALL CZCPSI ('OFFSET',0)
CALL CZCPSI ('SETPRO',IPROC)
CALL GSAVE(LUNITS(2),KEYS,NSAVE,IDEVT,IER)
CALL CZCPSR('OFFSET',IOFF)
NPHR = IOFF/ (4*CZBLK)

```

C Transfer output to file

```
CALL ACP_CALL (IPROC, QOUTPUT, ROUTBL, NPHR)
```

* We expect that the CONTROL process will execute the following commands
 * to terminate and close the FZ output file.

```

*        CALL ACP_JOB_INFO (1, OUTPUT, IPROC 1, IPROCN)
*        CALL CZCPSI ('OFFSET',0)
*        CALL CZCPSI ('SETPRO',IPROCL)
*        CALL GCLOSE(LUNITS(2), IERROR)
*        CALL CZCPSR('OFFSET',IOFF)
*        NPHR = IOFF/ (4*CZBLK)
*        CALL ACP_CALL(IPROCL,ACP$WAIT,ROUTBL,NPHR)
*        CALL ACP_CALL(IPROCL,ACP$WAIT,RFINIS)

```

3.6 Distribution of FFREAD input to WORKER processes

This example demonstrates distributed FFREAD input to a set of WORKER processes. The FFREAD data cards for the worker processes are written into a CPS memory buffer on the CONTROL process for access by the WORKER processes. Users should adjust the size of the memory buffer to match the requirements of their applications.

3.6.1 CONTROL process—server

C Define distributed FFREAD "logical unit".
 C Equate CPS block number to FFREAD "logical unit".

```

INTEGER FFLUN
PARAMETER ( FFLUN = 98 )

```

```

INTEGER CPSFF
PARAMETER ( CPSFF = 98 )

```

C Define FFREAD buffer size. Buffer is local.
 C Allow 50 card images.

```

INTEGER FFBUF
PARAMETER ( FFBUF = 1000 )

```

C Declare CPS memory buffer for distributed FFREAD input.

```

INTEGER MFFBUF
COMMON /CFFBUF/ MFFBUF(FFBUF)

```

```

C   Define block size (words) for distributed FFREAD input.
C   Use card images---FFREAD default value.

      INTEGER FFBLK
      PARAMETER ( FFBLK = 20 )

C   Declare character buffer for input.

      CHARACTER*(4*FFBLK) STRING

C   Initialize CPS software.

      CALL ACP_INIT

C   Declare CPS block for FFREAD input via CPS. CPS block is local.

      CALL ACP_DECLARE_BLOCK(MFFBUF,4*MFFBUF,CPSFF)

C   Transfer FFREAD cards for WORKER processes to local memory buffer.
C   Pack buffer with fixed-length Hollerith records.

      IFFOFF = 0
100  READ(FFLUN,'(A)',END=200) STRING
      CALL UCTOH(STRING,MFFBUF(1+IFFOFF),4,4*FFBLK)
      IFFOFF = IFFOFF + FFBLK
      GOTO 100
200  CONTINUE

C   Synchronize CONTROL and WORKER processes. WORKER processes may now
C   perform remote FFREAD input.

      CALL ACP_SYNC(ACP$ALL_PROCESSES,0)

3.6.2  WORKER process—client

C   Define distributed FFREAD "logical unit".
C   Equate CPS block number to FFREAD "logical unit".

      INTEGER FFLUN
      PARAMETER ( FFLUN = 98 )

      INTEGER CPSFF
      PARAMETER ( CPSFF = 98 )

C   Define FFREAD buffer size. Buffer is remote.
C   Allow 50 card images.

      INTEGER FFBUF
      PARAMETER ( FFBUF = 1000 )

C   External declaration for alternate input routine.

```

EXTERNAL FFCPS

C Initialize CPS software.

CALL ACP_INIT

C Obtain FFREAD data cards from memory buffer on CONTROL process.

C Set alternate input subroutine to allow FFREAD input via CPS.

CALL FFSET('LINP',-FFLUN)

CALL FFRSET(FFLUN,FFCPS)

C Define CPS block information for FFREAD input via CPS.

C CPS block is remote. Store CPS block number, buffer length (bytes) and

C CPS process number of CONTROL process.

CALL FFCPSI('CPSBUF',CPSFF)

CALL FFCPSI('BUFLN',4*FFBUF)

CALL ACP_JOB_INFO(1,CONTROL,IPROC1,IPROCN)

CALL FFCPSI('SETPRO',IPROC1)

C Synchronize CONTROL and WORKER processes to allow FFREAD cards to be

C written into memory buffer on CONTROL process.

CALL ACP_SYNC(ACP\$ALL_PROCESSES,0)

C Read data cards.

CALL FFCPSI('OFFSET',0)

CALL GFFGO

4.0 SUMMARY

CZ/CPS introduces recent developments in distributed computing techniques in the CERN Program Library into the CPS cooperative processes environment. CZ/CPS provides easy-to-use interprocess ZEBRA/FZ data communication for program initialization, event distribution, and event collection. The CPS ZEBRA Server provides a simple, yet familiar, interface for histogram distribution and collection via its integration of features from VAX/VMS global sections and RZ direct-access files. Distributed FFREAD input for CPS processes simplifies FFREAD initialization distribution by handling system and user-defined data cards on an equal basis.

A representative collection of examples demonstrates the capabilities of the CZ/CPS interface. Transfer of GEANT3 initialization data structures, GEANT3 event FZ output, HBOOK4 histogram input, output and summation, and distribution of FFREAD data cards can be easily accomplished.

REFERENCES

1. *ACP Cooperative Processes User's Manual*, Fermilab Computer Research & Development Department, Fermi National Accelerator Laboratory.
2. CERN Program Library manual, G. Benassi, ed., CERN Computer Centre.
3. R. Brun, M. Goossens, and J. Zoll, *ZEBRA User Guide*, CERN Computer Centre Program Library Long Writeup Q100.
4. J. Zoll, *ZEBRA Reference Manual*, CERN Computer Centre.
5. R. Brun and D. Lienart, *HBOOK User Guide, Version 4*, CERN Computer Centre Program Library Long Writeup Y250.
6. R. Brun, B. Holl, J. Shiers, O. Couet, C. Magnin, J. Wood, F. Hemmer, and B. Segal, *CSPACK*, CERN Computer Centre Program Library Q124.
7. R. Brun, F. Bruyant, M. Maire, A. C. McPherson, and P. Zancarini, *GEANT3 User's Guide*, CERN DD/EE/84-1, May 1986.
8. R. Brun, R. Hagelberg, J. C. Lassalle, G. Misuri, and J. Vobruuggen, *FFREAD User Guide and Reference Manual*, CERN Computer Centre Program Library Long Writeup I302.
9. L. A. Roberts, *BCD/CPS, An event-level GEANT3 parallelization via CPS*, SSCL-413, SSC Laboratory.
10. F. E. Paige and S. D. Protopopescu, *ISAJET 6.36*, Brookhaven National Laboratory.
11. H.-U. Bengtsson and T. Sjöstrand, *PYTHIA 5.5*, University of Lund and CERN/TH.