A User's Guide for the PASSION Runtime Library Version 1.0[1]

A. Choudhary *R. Bordawekar* *S. More* *K. Sivaram* *R. Thakur*

Syracuse University, Syracuse, NY 13244

`passion@cat.syr.edu`

NPAC Technical Report SCCS–702, February 1995

[1]This work was supported in part by NSF Young Investigator Award CCR-9357840, grants from Intel SSD and IBM Corp., and in part by USRA CESDIS Contract # 5555-26. This work was performed in part using the Intel Touchstone Delta and Paragon Systems operated by Caltech on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by CRPC.

Contents

List of Figures

Chapter 1

Introduction

The PASSION Runtime Library provides software support for high-performance parallel I/O on distributed memory parallel computers. PASSION runtime routines can be used to efficiently perform the I/O required in out-of-core programs. PASSION assumes a loosely synchronous Single Program Multiple Data (SPMD) model of parallel computation. It provides the user with a simple high-level interface, which is a level higher than any of the existing parallel file system interfaces or even the proposed MPI-IO interface [CFH+94]. For example, the user only needs to specify what section of the array needs to be read in terms of its lower-bound, upper-bound and stride in each dimension, and the PASSION Runtime Library will fetch it in an efficient manner. A number of optimizations, such as Data Sieving, Data Prefetching, Data Reuse, and the Extended Two-Phase Method, have been incorporated in the library for improved performance [TBC+94b, TBC94a, TC95]. The library can either be used directly by application programmers, or a compiler could translate out-of-core programs written in a high-level data-parallel language like HPF to node programs with calls to the runtime library for I/O. Further details about the PASSION project can be found in [CBH+94, TBC+94b, TBC94a, TC95, BC94, BdRC93, dRHC94, SC94, dRBC93].

1.1 Features of Version 1.0

Version 1.0 of the PASSION Runtime Library provides a number of routines for parallel access to data in files. It supports two models for storing and accessing data, called the Local Placement Model (LPM) and the Global Placement Model (GPM) [TBC94a, CBH+94], which are explained in Chapter 2. Version 1.0 of PASSION can be used on the Intel Paragon, Touchstone Delta and iPSC/860 systems. The PASSION routines can currently be called only from C programs. This version supports only two-dimensional arrays stored in the file in row major order. In future versions, we plan to provide support for Fortran programs, allow any dimensional arrays stored in the file in any order, and port the Runtime Library to the IBM SP-2.

Chapter 2

Data Access Models

The PASSION Runtime Library supports two models for storing and accessing data, called the Local Placement Model (LPM) and the Global Placement Model (GPM) [TBC94a, CBH$^+$94]. In all cases, the files are unformatted binary files.

2.1 Local Placement Model (LPM)

In the Local Placement Model (Figure 2.1), the local array of each processor is stored in a separate file. Thus, for each array, there are as many files as the number of processors. A processor cannot directly access data from the local array files of other processors.

2.2 Global Placement Model (GPM)

In the Global Placement Model (GPM) (Figure 2.1), the entire array is stored in a single file. Each processor can directly access any portion of the file.

2.3 Terminology

The definitions of some key terms used throughout this document are given below:-

1. **Local Array File (LAF):** In the Local Placement Model, the files in which the local arrays of processors are stored are called *Local Array Files (LAFs).*
2. **Global Array File (GAF):** In the Global Placement Model, the file in which the global array is stored is called *Global File.*
3. **Out-of-Core Local Array (OCLA):** The portion of the array stored in the LAF is called *Out of Core Local Array.*
4. **In-core Local Array (ICLA):** The portion of the OCLA which is present in the main memory of a processor and is used for computations is called *In Core Local Array.*
5. **Overlap Area:** It is used to store the off-processor data that is needed by each processor. The processor should treat it as read-only since it does not logically belong to the processor.

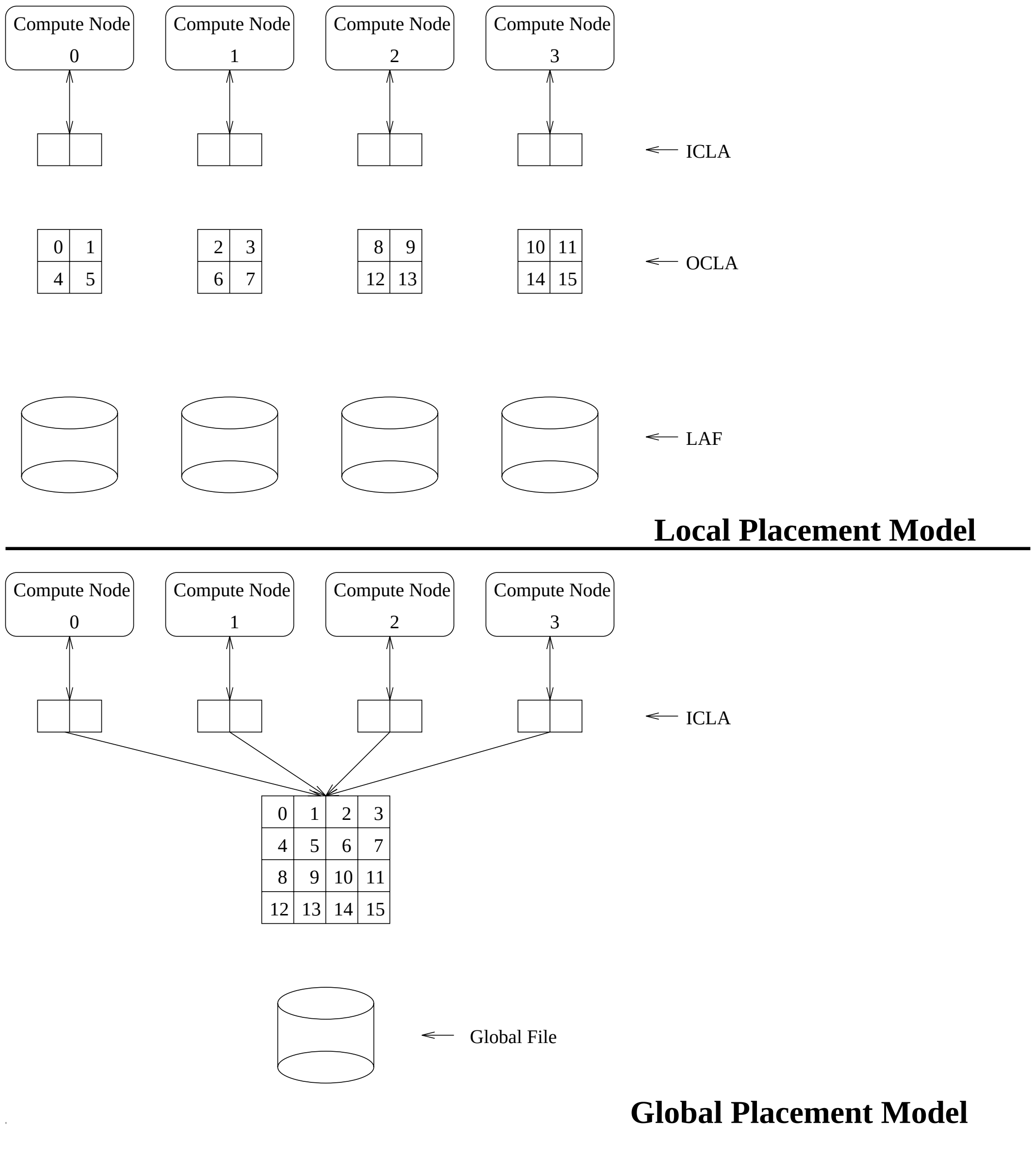


Figure 2.1: Data Organization in the Local and Global Placement Models

Chapter 3

Using the PASSION Data Structures

The various data structures used in the PASSION library are described below.

3.1 Out-of-Core Array Descriptor (OCAD)

Each out-of-core array has a descriptor associated with it called the Out-of-Core Array Descriptor (OCAD). The OCAD contains the following information about the array

- Number of dimensions
- Size of the global array
- Size of each element of the array in bytes
- Number of processors in each dimension
- Distribution of the array in each dimension
- Size of the ICLA
- Size of the overlap area
- Size of the OCLA

This structure should be allocated and deallocated using the library routines *PASSION_malloc_OCAD* and *PASSION_free_OCAD* respectively. *The user is responsible for allocating and deallocating the OCAD and also for providing the information to be stored in the OCAD.* The routine *PASSION_fill_OCAD* should be used to fill in the OCAD.

3.2 Parallel File Pointer (PFILE)

This structure contains the following information about the parallel file :

- System file descriptor
- Header size

This structure is allocated and initialized by the *PASSION_open* routine, and deallocated by the *PASSION_close* routine. It acts as a file pointer for the parallel file and should be passed as an argument to all PASSION routines which access the file.

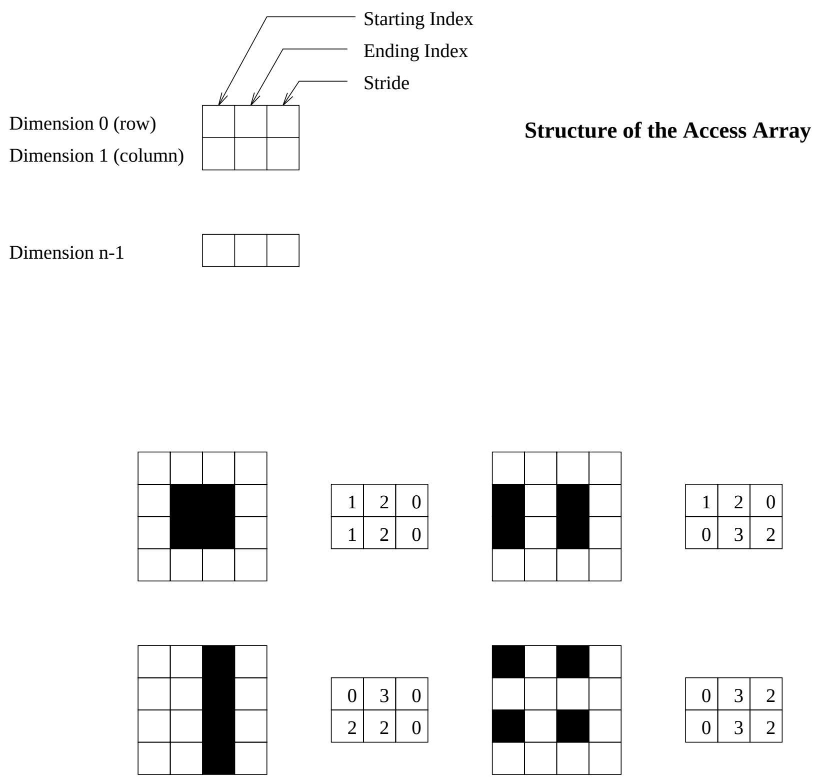


Figure 3.1: Specifying Array Sections

3.3 Prefetch Descriptor (PREFETCH)

This data structure is used to store information regarding pending prefetch read operation(s). It stores the ids returned by *PASSION_read_prefetch*. It is allocated by the *PASSION_read_prefetch* routine and is deallocated by the *PASSION_prefetch_wait* routine.

3.4 Reuse Descriptor (REUSE)

This data structure is used to implement the data reuse operation. It is allocated by the *PASSION_reuse_init* routine and is deallocated by the *PASSION_read_reuse* routine.

3.5 Access Array

This data structure is used to specify which section of the array needs to be read or written. It is a two dimensional array of size *No. of dimensions of the out-of-core array* $\times 3$. Row i contains access information about dimension i of the out-of-core array, in the form of its lower bound, upper bound and stride. (See Figure 3.1). *The user has to provide this array to the PASSION read/write routines.*

Chapter 4

Defining an Array

4.1 Defining the Array Parameters

To define an array, the user first needs to define :

Number of dimensions of the array

Currently only two dimensional arrays are supported. One way to specify the number of dimensions in the program is :

```
#define DIMENSIONS ((int)2)
```

Storage type

This specifies how the array is stored in the file, i.e. row-major or column-major. *Currently only row-major storage is supported.*

Size of the array

The size of the array is specified using a one dimensional array of integers. It contains as many elements as the number of dimensions in the out-of-core array. For example, the size of a 1024×1024 array can be specified as follows:-

```
#define DIMENSIONS ((int)2)
int Size[DIMENSIONS]={1024, 1024};
```

Number of processors in each dimension

The number of processors over which each dimension of the array is distributed is specified using a one dimensional array. For example, a two dimensional out-of-core array with rows distributed among 4 processors and the columns collapsed can be specified as follows:-

```
#define DIMENSIONS ((int)2)
int Procs[DIMENSIONS]={4, 1};
```

Array Distribution

The distribution of the array is specified using a two dimensional array of integers. The size of this array is *no. of dimensions of the out_of_core array* $\times 2$. Each row of this array specifies the distribution in that dimension.

Distribution	**Comments**
NO_DISTRIBUTION	No. of processors in this dimension must be one
BLOCK_DISTRIBUTION	Each processor gets a block of elements in that dimension
CYCLIC_DISTRIBUTION	Each processor gets every p^{th} element where p is the number of processors in that dimension

Table 4.1: Array Distributions

The first element in the row specifies the kind of distribution and the second element in the row specifies the *block size* if the distribution is *cyclic.* A column block distribution of a two dimensional array can be specified as follows :

```
#include "passion.h"

#define DIMENSIONS ((int)2)
int Distribution[DIMENSIONS][2]={{NO_DISTRIBUTION, 0},
                                 {BLOCK_DISTRIBUTION, 0}
                                };
```

Size of the OCLA

The size of the OCLA is defined using a one dimensional array of integers. It contains as many elements as the number of dimensions in the out-of-core array. For example, the OCLA size for a 1024×1024 size array distributed in a *row block* fashion over four processors can be specified as follows.

```
#define DIMENSIONS ((int)2)
int OCLA_size[DIMENSIONS]={256, 1024};
```

Size of the ICLA

The size of ICLA is specified using a one dimensional array of integers. It contains as many elements as the number of dimensions in the out-of-core array. The ICLA size depends on the amount of main memory available and is usually less than the size of the OCLA. For example, if the OCLA size is 256×1024, but only 4×1024 can be stored in-core, the ICLA can be specified as :

```
#define DIMENSIONS ((int)2)
int ICLA_size[DIMENSIONS]={4, 1024};
```

Overlap information

The size of the overlap area is specified using a two dimensional array of size equal to *the no. of dimensions of the out-of-core array* $\times 2$. Each row contains the overlap information in that dimension. The first element gives the lower overlap area and the second element gives the upper overlap area. If the overlap area is one row in either direction, it can be specified as :

```
#define DIMENSIONS ((int)2)
int overlap_size[DIMENSIONS][2]={{1, 1}, {0, 0}};
```

Size of each element in the array

The size of each element is an integer giving the size in bytes. For example, the element size for an array of doubles can be specified as :

```
int ElementSize = sizeof(double);
```

All this information needs to be passed to the routine *PASSION_fill_OCAD*, which initializes the OCAD.

4.2 Creating and Initializing the OCAD

All PASSION routines require a pointer to the OCAD. The array information is stored inside the OCAD. The OCAD can be created and initialized as follows :

Define the array parameters

This can be done as explained in Section 4.1.

Allocate the OCAD

The *PASSION_malloc_OCAD* routine needs to be used to allocate the OCAD.

Fill in the OCAD

The *PASSION_fill_OCAD* function is used to fill the OCAD. It accepts the pointer to the newly allocated OCAD and other array parameters (explained in Section 4.1).

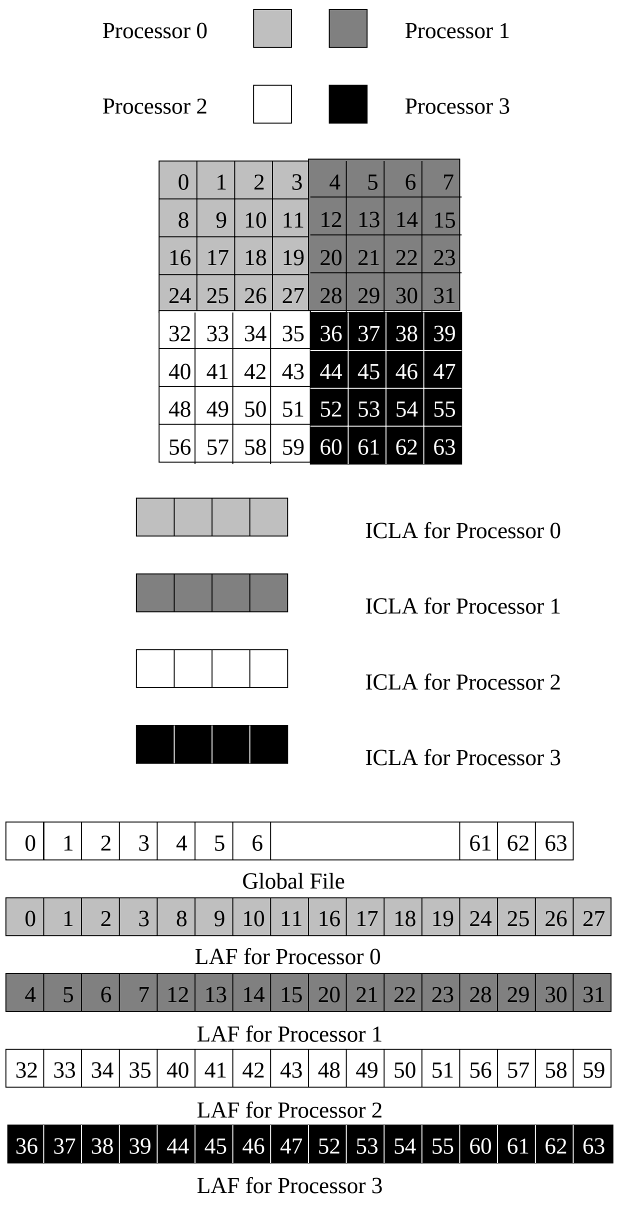
Access the array

The parallel file(s) can now be accessed (explained in Chapter 5).

Deallocate the OCAD

After all the computation on the array is done, the OCAD can be deallocated using *PASSION_free_OCAD*.

The entire process is illustrated in Figure 4.1. It is assumed in the figure that there is no overlap area. Refer to Figure 4.2 to see how to define an array with overlap area.



```
#include "passion.h"

#define DIMENSIONS 2
#define NUM_ROWS 8
#define NUM_COLS 8
#define PROCS_DIM_0 2
#define PROCS_DIM_1 2

#define OCLA_DIM_0 (NUM_ROWS/PROCS_DIM_0)
#define OCLA_DIM_1 (NUM_COLS/PROCS_DIM_1)
#define ICLA_DIM_0 1
#define ICLA_DIM_1 4
int Size[DIMENSIONS]={NUM_ROWS,
                                NUM_COLS};
int Procs[DIMENSIONS]=
             {PROCS_DIM_0, PROCS_DIM_1};
int Distribution[DIMENSIONS][2]=
                  {{BLOCK_DISTRIBUTION, 0},
                   {BLOCK_DISTRIBUTION, 0}};
int OCCLA_size[DIMENSIONS]=
                      {OCLA_DIM_0, OCLA_DIM_1};
int ICLA_size[DIMENSIONS]=
                      {ICLA_DIM_0, ICLA_DIM_1};
int overlap_info[DIMENSIONS][2]=
                {{UP_OVERLAP, DOWN_OVERLAP),
                 {LEFT_OVERLAP, RIGHT_OVERLAP}};
int ElemSize = sizeof(double);
OCAD *OCADp;


OCADp = mallocOCAD(DIMENSIONS, ROW_MAJOR);


PASSION_ArrayInfo( OCADp,  Size,  Distribution, Procs,
OCLA_size, ICLA_size,  overlap_info, ElemSize);
```

Figure 4.1: Creating the OCAD

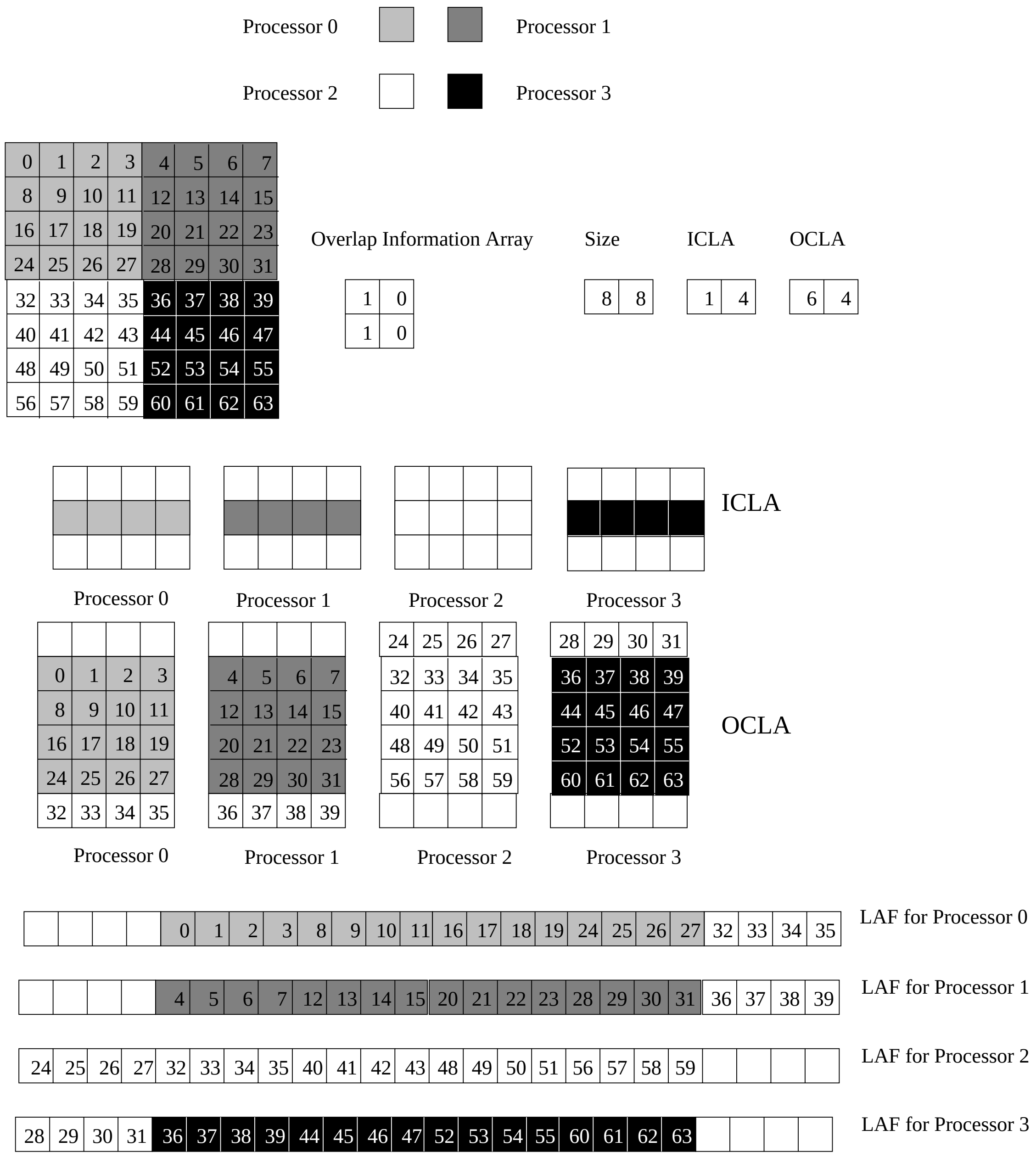


Figure 4.2: Specifying Overlap Information

Chapter 5

Accessing the Array

5.1 Opening the Array File

The *PASSION_open* routine can be used to open array file(s). Note that in the Global Placement Model, all processors open the same file, whereas in the Local Placement Model each processor opens a separate file. *PASSION_open* accepts as parameters the name of the file and size of the header at the start of the file (if any).

5.1.1 Opening a File in the Local Placement Model

If there are n processors and *FILE* is the generic file name, the individual files can be named *FILE0, FILE1,* ... *FILEn.*

```
#include "passion.h"

#define GENERIC_NAME "FILE"
#define HEADER_SIZE  ((int)1024)

PFILE *PFilePtr;
char FileName[10];

  sprintf(FileName, "%s%d", GENERIC_NAME, mynode());

  PFilePtr = PASSION_open(FileName, HEADER_SIZE);
  if (PFilePtr == (PFILE *)0)
  {
    printf("Error opening the parallel file.");
    /* exit */
  }
```

5.1.2 Opening a File in the Global Placement Model

Let the name of the global file be *FILE.*

```
#include "passion.h"

#define FILE_NAME "FILE"
#define HEADER_SIZE  ((int)1024)
```

```
PFILE *PFilePtr;

  PFilePtr = PASSION_open(FILE_NAME, HEADER_SIZE);
  if (PFilePtr == (PFILE *)0)
  {
    printf("Error opening the parallel file.");
    /* exit */
  }
```

5.2 Closing the Array File

The *PASSION_close* routine can be used to close the array file(s). It accepts a pointer to the PFILE structure as a parameter. The pointer should be the one returned by an earlier *PASSION_open* routine.

```
#include "passion.h"

PFILE *PFilePtr;

/* Code to open the file and do something with the data */

  PASSION_close(PFilePtr);
```

5.3 Reading the Array File

A variety of PASSION routines are provided to read data from the files efficiently. The simplest routine to read data is *PASSION_read*. It reads the entire OCLA into the ICLA. Note that the ICLA size should be equal to the OCLA size to use this routine. This routine should be used in the Local Placement Model only.

```
#include "passion.h"

/* define OCLA size */

#define ICLA_DIM1 OCLA_DIM1
#define ICLA_DIM2 OCLA_DIM2

PFILE *PFilePtr;
OCAD  *OCADp;
double Array[ICLA_DIM1][ICLA_DIM2];

  /* Fill up the OCAD */

  /* Open the file */

  if (PASSION_read(PFilePtr, OCADp, (char *)Array) != 0)
  {
    printf("Error reading the parallel file");
  }
  /* Use the data */
  /* Close the file */
```

5.3.1 Reading Array Sections

PASSION provides routines to read sections of the array with strides in each dimension. Separate routines are provided for reading array sections in the Local and Global Placement Models.

Local Placement Model

The routine *PASSION_read_section* is used to read array sections in the Local Placement Model. The Data Sieving Method described in [TBC+94b, CBH+94] is used for better performance. The array section is specified using the *access array* (See Section 3.5 for details about how to specify array sections). This routine reads the array section from the OCLA to the specified position. The shape of the section is retained. Also, the section is stored without stride in the ICLA, even if there was a stride in the OCLA. This is done in order to save memory.

```
#include "passion.h"

/* define ICLA size */

PFILE *PFilePtr;
OCAD  *OCADp;
double Array[ICLA_DIM1][ICLA_DIM2];
int AccessArray[DIMENSIONS][3], i, j;

  /* Fill up the OCAD */

  /* Open the file */

  /* fill up the access array and set i & j to the location in the
  ICLA where the section is to be read */

  if (PASSION_read_section(PFilePtr, OCADp, (char *)Array, i, j,
                           AccessArray) != 0)
  {
    printf("Error reading the parallel file");
  }

  /* Use the data */

  /* Close the file */
```

Global Placement Model

The routine *PASSION_global_read* is used to read array sections in the Global Placement Model. Since this is a global file, note that all processors should specify the same file. Each processor can access any arbitrary section of the array. The sections requested by the processors could be distinct, overlapping or even identical. This routine reads the sections in an efficient manner using the Extended Two-Phase Method, which was proposed in [TC95]. The array section is specified using the *access array* (See Section 3.5 for details about how to specify array sections). The routine reads the array section from the global array file into the specified location in main memory. The shape of the section is retained. Also, the section is stored without stride in main memory, even if there was a stride specified in the out-of-core global array. This routine uses *collective I/O*, so all processors must call the routine. Even if a processor does not want to read any data, it must call the routine and specify an empty section. (See Figure 5.1).

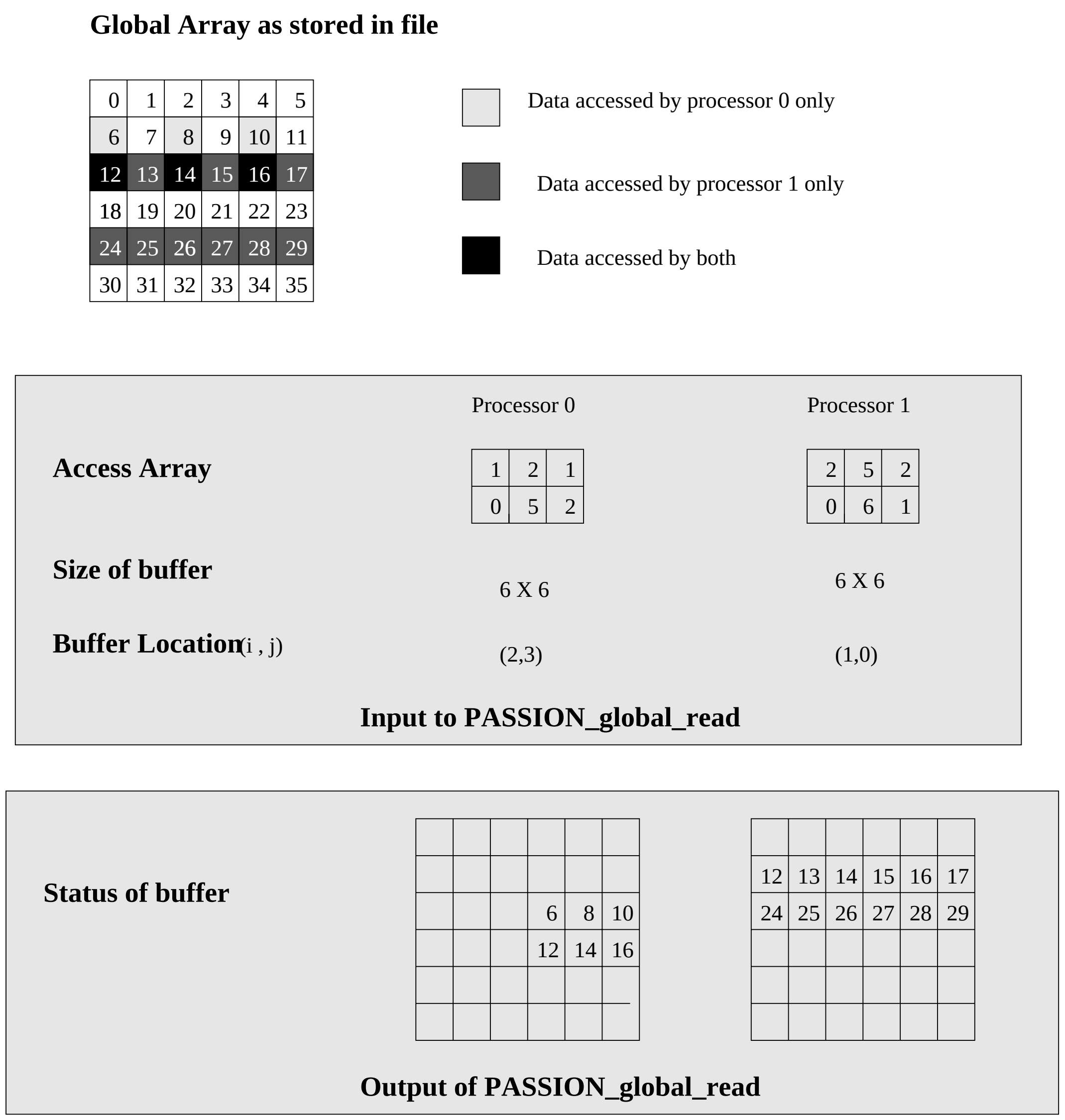


Figure 5.1: Global Read Operation

```
#include "passion.h"

/* define the buffer size */

PFILE *PFilePtr;
OCAD  *OCADp;
double Array[GLOBAL_ROWS][GLOBAL_COLS];
int AccessArray[DIMENSIONS][3], i, j;
int nprocs = NPROCS;

  /* Fill up the OCAD */

  /* Open the file */

  /* fill up the access array and set i & j to the location in
   the in-core array  where the section is to be read */

  if (PASSION_global_read(PFilePtr, OCADp, (char *)Array, i, j,
                          AccessArray,nprocs) != 0)
  {
    printf("Error reading the parallel file");
  }

  /* Use the data */

  /* Close the file */
```

5.3.2 Using Prefetching for Faster Access

Routines are provided for prefetching data before it is needed, in order to reduce I/O time.[1] Data prefetching can be used to overlap computation and communication with I/O and is described in [TBC$^+$94b]. Prefetching can be used in the following manner :

```
  read the first section of  data from the
  OCLA  to the ICLA (PASSION_read_section)
  while there is more data to read
    issue a prefetch read for the next section (PASSION_read_prefetch)
    process the section currently in memory
    wait for the prefetch read to get over (PASSION_prefetch_wait)
  endwhile
```

The process is illustrated in Figure 5.2. *Note that currently prefetching can only be used for sections with stride in a column, but not stride in a row.*

5.3.3 Data Reuse

Very often, some of the data from the current section in main memory is also needed for the computation on the next section. Data reuse is an optimization which reuses data already present in main memory, instead of reading it again from disk, and is described in [TBC$^+$94b]. PASSION provides two routines *PASSION_reuse_init* and *PASSION_read_reuse* for reusing data. Data reuse is currently supported only in

[1] Currently available only for the Local Placement Model.

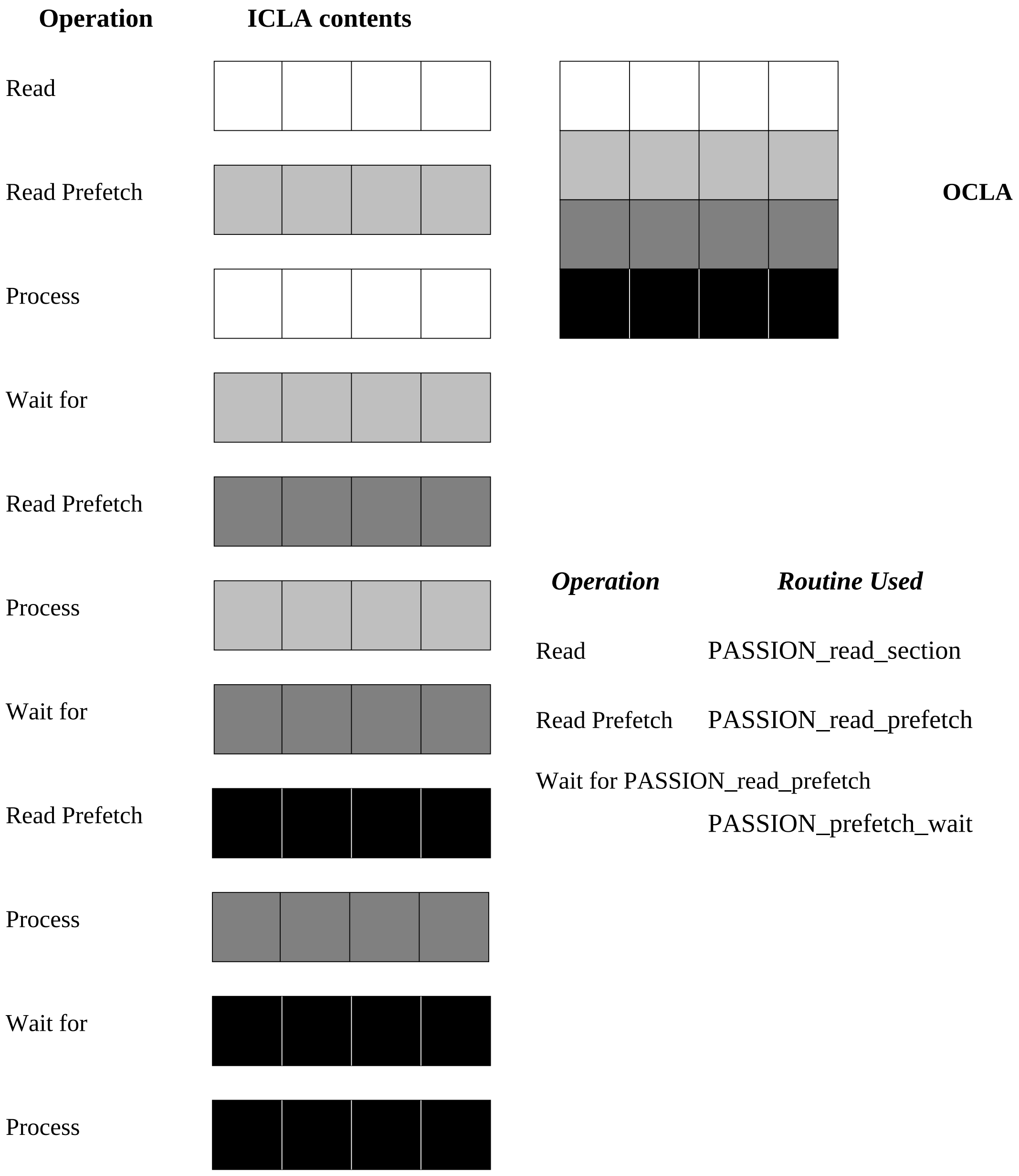


Figure 5.2: Using Prefetching to Overlap Computation and Communication with I/O

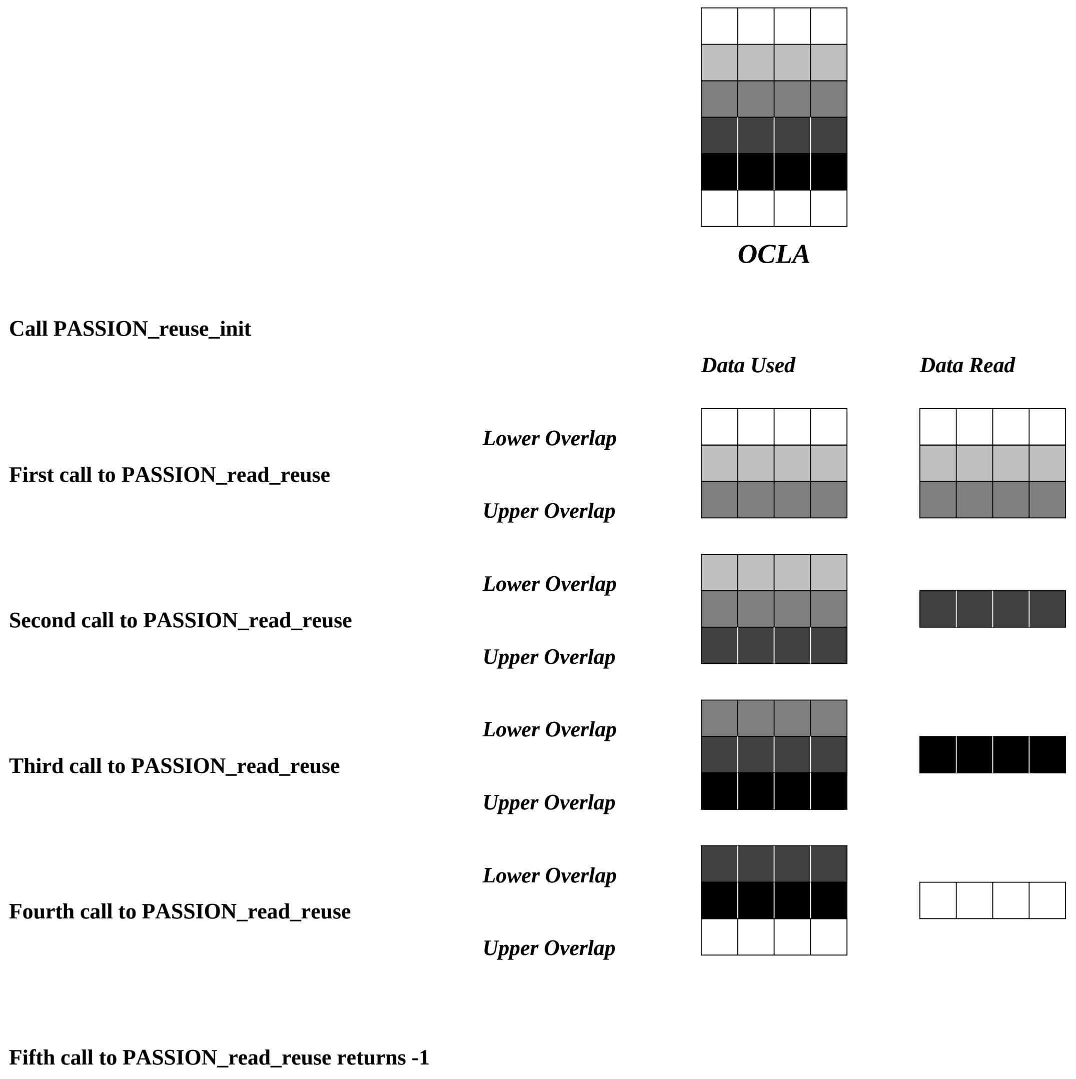

OCLA
Call PASSION_reuse_init
Data Used
Data Read
Lower Overlap
First call to PASSION_read_reuse
Upper Overlap
Lower Overlap
Second call to PASSION_read_reuse
Upper Overlap
Lower Overlap
Third call to PASSION_read_reuse
Upper Overlap
Lower Overlap
Fourth call to PASSION_read_reuse
Upper Overlap
Fifth call to PASSION_read_reuse returns -1

Figure 5.3: Data Reuse

the Local Placement Model, when each section read is a slab of rows with overlap area at the top and bottom. The reuse routines can be used as follows :

```
initialize the reuse operation (PASSION_reuse\_init)
while there is more data (PASSION_read_reuse)
  process the data
endwhile
```

PASSION_reuse_init initializes the REUSE data structure (See Section 3.4) and returns a pointer to this structure. *PASSION_read_reuse* is used to actually read the data. This routine determines the size of the overlap area from the OCAD. Suppose the overlap area is x rows at the top and bottom. Then for the first read, *PASSION_read_reuse* reads the entire section from the file. For subsequent reads, it moves x rows from the lower overlap area to the beginning of the ICLA and x rows from the end of the ICLA to the upper overlap area, and only the remaining rows are read from the file. Figure 5.3 illustrates how data reuse is performed. An example program is given in Section 6.6.

5.4 Writing the Array File

A variety of PASSION routines are provided to write data to files efficiently. The simplest routine to write data is *PASSION_write*. It writes the entire ICLA to the OCLA. Note that the ICLA size should be equal to the OCLA size to use this routine. This routine should be used in the Local Placement Model only.

```
#include "passion.h"

/* define OCLA size */

#define ICLA_DIM1 OCLA_DIM1
#define ICLA_DIM2 OCLA_DIM2

PFILE *PFilePtr;
OCAD  *OCADp;
double Array[ICLA_DIM1][ICLA_DIM2];

  /* Fill up the OCAD */

  /* Open the file */

  if (PASSION_write(PFilePtr, OCADp, (char *)Array) != 0)
  {
    printf("Error writing the parallel file");
  }

  /* Close the file */
```

5.4.1 Writing Array Sections

PASSION provides routines to write sections of the array to the file, with strides in each dimension. Separate routines are provided for writing array sections in the Local and Global Placement Models

Local Placement Model

The routine *PASSION_write_section* is used to write array sections in the Local Placement Model. The Data Sieving Method described in [TBC+94b, CBH+94] is used for better performance. The array section is specified using the *access array* (See Section 3.5 for details about how to specify array sections). This routine writes the array section from the specified location in the ICLA to the OCLA. The shape of the section is retained. Also, the section is assumed to be stored without stride in the ICLA, but is written with the specified stride in the OCLA.

```
#include "passion.h"

/* define ICLA size */

PFILE *PFilePtr;
OCAD  *OCADp;
double Array[ICLA_DIM1][ICLA_DIM2];
int AccessArray[DIMENSIONS][3], i, j;

  /* Fill up the OCAD */
  /* Open the file */

  /* fill up the access array and set i & j to the location in
  ICLA from where the section is to be taken */

  if (PASSION_write_section(PFilePtr, OCADp, (char *)Array, i, j,
                           AccessArray) != 0)
  {
    printf("Error writing the parallel file");
  }

  /* Use the data */

  /* Close the file */
```

Global Placement Model

The routine *PASSION_global_write* is used to write array sections in the Global Placement Model. Since data is to be written to a global file, all processors should specify the same file name. It is assumed that each processor writes a distinct section of the file. The Extended Two-Phase Method [TC95] is used to write sections efficiently. The array section is specified using the *access array* (See Section 3.5 for details about how to specify array sections). This routine writes the array section to the global file from the specified location in main memory. The shape of the section is retained. Also, the section is assumed to be stored without stride in main memory, but is written with the specified stride in the global array file. This routine uses *collective I/O*, so all processors must call the routine. Even if a processor does not want to write data, it must call the routine and specify an empty section.

```
#include "passion.h"
/* define the global array  size */
PFILE *PFilePtr;
OCAD  *OCADp;
double Array[GLOBAL_ROWS][GLOBAL_COLS];
int AccessArray[DIMENSIONS][3], i, j;
int nprocs = NPROCS;
  /* Fill up the OCAD */
  /* Open the file */
  /* fill up the access array and set i & j to the location in
  the in-core array from where the section is to be taken */
  if (PASSION_global_write(PFilePtr, OCADp, (char *)Array, i, j,
                           AccessArray,nprocs) != 0)
  {
    printf("Error writing the parallel file");
  }
  /* Use the data */
  /* Close the file */
```

Chapter 6

Example Programs

Some simple programs which use the PASSION runtime routines for I/O are given in this chapter. Programs in Sections 6.1, 6.2, 6.3, 6.4, 6.5 and 6.6 are for the Local Placement Model. Programs in Sections 6.7 and 6.8 are for the Global Placement Model. A sample makefile is given in Section 6.9.

6.1 PROGRAM 1 : Creating Local Array Files

```
/* This program creates local array files using PASSION_write */

#include <string.h>
#include <fcntl.h>

#include "passion.h"

#define HEADER_STRING "PASSION File Header"    /* some header */

#define DIMENSIONS (unsigned int)2

void main(int argc, char **argv)
{
PFILE *PFilePtr;
int global_size[DIMENSIONS] = {16, 16};
int nprocs[DIMENSIONS] = {2, 1};
int distribution[DIMENSIONS][2] = {{BLOCK_DISTRIBUTION, -1},
                                   {NO_DISTRIBUTION, -1}
                                  };
int overlap[DIMENSIONS][2] = {{0, 0}, {0, 0}};
int icla_size[DIMENSIONS][2] = {{0, 7}, {0, 15}};
int ocla_size[DIMENSIONS] = {8,16};
OCAD *OCADp;
double Array[8][16];
char FileName[20];
int IclaSize;
int i, j, MyNode=mynode();

  sprintf(FileName, "FILE%d", MyNode);
  close(open(FileName,O_CREAT|O_RDWR|O_TRUNC,0600));

/* create and fill up OCAD with the above defined information */

  OCADp = PASSION_malloc_OCAD(DIMENSIONS, ROW_MAJOR);
  if (OCADp == (OCAD *)0)
  {
    printf("Error in OCAD.\n");
    exit(0);
  }

  if (PASSION_fill_OCAD(OCADp, global_size, distribution, nprocs, ocla_size,
          icla_size, overlap, sizeof(double)) != 0)
  {
    printf("Error in filling OCAD.\n");
    exit(0);
  }

  if ((PFilePtr = PASSION_open(FileName, strlen(HEADER_STRING)+1)) == (PFILE *)0)
  {
    printf("Error in opening the file.\n");
    exit(0);
  }

  if (PASSION_write_header(PFilePtr, HEADER_STRING) != 0)
  {
    printf("Cannot write header.\n");
    exit(0);
  }

  for (i = 0; i < OCADp->ocla_size[0]; i++)
     for (j = 0; j < OCADp->ocla_size[1]; j++)
        Array[i][j] = (double)((i * OCADp->ocla_size[1]) + j);


  if (PASSION_write(PFilePtr, OCADp, (char *)Array) != 0)
  {
    printf("Error in PASSION_write.\n");
    exit(0);
  }

  if (PASSION_close(PFilePtr) == -1)
  {
    printf("Error in closing the file.\n");
    exit(0);
  }

  PASSION_free_OCAD(OCADp);
}
```

6.2 PROGRAM 2 : Reading Local Array Files

```
/* This program reads and displays an entire local array file using
   PASSION_read */

#include <string.h>
#include "passion.h"

#define DIMENSIONS (unsigned int)2
#define HEADER_STRING "PASSION File Header"

void main(int argc, char **argv)
{
PFILE *PFilePtr;
int global_size[DIMENSIONS] = {16, 16};
int nprocs[DIMENSIONS] = {2, 1};
int distribution[DIMENSIONS][2] = {{BLOCK_DISTRIBUTION, -1},
                                   {NO_DISTRIBUTION, -1}
                                  };
int overlap[DIMENSIONS][2] = {{0, 0}, {0, 0}};
int icla_size[DIMENSIONS][2] = {{0, 7}, {0, 15}};
int ocla_size[DIMENSIONS] = {8, 16};
OCAD *OCADp;
double Array[8][16];
char FileName[20], header[50];
int IclaSize;
int i, j, k, MyNode=mynode();

  sprintf(FileName, "FILE%d", mynode());

  OCADp = PASSION_malloc_OCAD(DIMENSIONS,ROW_MAJOR);
  if (OCADp == (OCAD *)0)
  {
    printf("Error in OCAD.\n");
    exit(0);
  }
  if (PASSION_fill_OCAD(OCADp, global_size, distribution, nprocs, ocla_size,
         icla_size, overlap, sizeof(double)) != 0)
  {
    printf("Error in filling OCAD.\n");
    exit(0);
  }

  if ((PFilePtr = PASSION_open(FileName, strlen(HEADER_STRING)+1)) == (PFILE *)0)
  {
    printf("Error in opening the file.\n");
    exit(0);
  }


  if (PASSION_read_header(PFilePtr, (char *) header) != 0)
  {
    printf("Cannot read header.\n");
    exit(0);
  }

  printf("Header => %s \n", header);


  if (PASSION_read(PFilePtr, OCADp, (char *)Array) != 0)
  {
     printf("Error in PASSION_read.\n");
     exit(0);
  }

  if (PASSION_close(PFilePtr) == -1)
  {
    printf("Error in closing the file.\n");
    exit(0);
  }

  for (k=0; k<numnodes(); k++)
  {
    if (k == mynode()) {
      for (i = 0; i < OCADp->ocla_size[0]; i++)
      {
  printf("[%d]", MyNode);
  for (j = 0; j < OCADp->ocla_size[1]; j++)
    printf("%.0f\t", (float)(Array[i][j]));
  printf("\n");
      }
      gsync();
    }
  }

  PASSION_free_OCAD(OCADp);
}
```

6.3 PROGRAM 3 : Reading Array Sections in the Local Placement Model

```
/* This program reads a section of the OCLA using PASSION_read_section */

#include "passion.h"

#define DIMENSIONS (unsigned int)2
#define HEADER_STRING "PASSION File Header"

void main(int argc, char **argv)
{
```

```
PFILE *PFilePtr;
int global_size[DIMENSIONS] = {16, 16};
int nprocs[DIMENSIONS] = {2, 1};
int distribution[DIMENSIONS][2] = {{BLOCK_DISTRIBUTION, -1},
                                   {NO_DISTRIBUTION, -1}
                                  };
int AccessArray[DIMENSIONS][3];
int overlap[DIMENSIONS][2] = {{0, 0}, {0, 0}};
int icla_size[DIMENSIONS][2] = {{0, 7}, {0, 15}};
int ocla_size[DIMENSIONS] = {8,16};
OCAD *OCADp;
double *Array;
char FileName[20];
int IclaSize;
int i, j, MyNode=mynode();

  sprintf(FileName, "FILE%d", mynode());

  OCADp = PASSION_malloc_OCAD(DIMENSIONS, ROW_MAJOR);
  if (OCADp == (OCAD *)0)
  {
    printf("Error in OCAD.\n");
    exit(0);
  }
  if (PASSION_fill_OCAD(OCADp, global_size, distribution, nprocs, ocla_size,
         icla_size, overlap, sizeof(double)) != 0)
  {
    printf("Error in filling OCAD.\n");
    exit(0);
  }

  if ((PFilePtr = PASSION_open(FileName, strlen(HEADER_STRING)+1)) == (PFILE *)0)
  {
    printf("Error in opening the file.\n");
    return;
  }

  IclaSize = (icla_size[0][1] - icla_size[0][0] + 1) *
     (icla_size[1][1] - icla_size[1][0] + 1);
  Array = (double *)malloc(IclaSize * sizeof(double));

/* specify the access pattern */

  AccessArray[0][0] = 0;
  AccessArray[0][1] = icla_size[0][1] - icla_size[0][0];
  AccessArray[0][2] = 2;
  AccessArray[1][0] = 0;
  AccessArray[1][1] = icla_size[1][1] - icla_size[1][0];
  AccessArray[1][2] = 2;

  if (PASSION_read_section(PFilePtr, OCADp, (char *)Array, 0, 0,
      AccessArray) != 0)
  {
      printf("Error in PASSION_read_section.\n");
      exit(0);
  }

  for (j=0; j<numnodes(); j++)
    {
      if (MyNode == j)
  {
    printf("[%d]", MyNode);
    for (i = 0; i < IclaSize; i++)
      {
        printf("%.2f\t", (float)*(Array + i));
      }
    printf("\n");
  }
      gsync();
    }

  if (PASSION_close(PFilePtr) == -1)
  {
    printf("Error in closing the file.\n");
    exit(0);
  }

  PASSION_free_OCAD(OCADp);
}
```

6.4 PROGRAM 4 : Writing Array Sections in the Local Placement Model

```
/* This program writes a section from the ICLA to the local array file
   using PASSION_write_section */

#include "passion.h"

#define DIMENSIONS (unsigned int)2
#define HEADER_STRING "PASSION File Header"

void main(int argc, char **argv)
{
PFILE *PFilePtr;
int global_size[DIMENSIONS] = {16, 16};
int nprocs[DIMENSIONS] = {2, 1};
int distribution[DIMENSIONS][2] = {{BLOCK_DISTRIBUTION, -1},
                                    {NO_DISTRIBUTION, -1}
                                   };
int AccessArray[DIMENSIONS][3];
int overlap[DIMENSIONS][2] = {{0, 0}, {0, 0}};
int icla_size[DIMENSIONS][2] = {{0, 7}, {0, 15}};
int ocla_size[DIMENSIONS] = {8,16};
OCAD *OCADp;
double *Array, count;
char FileName[20];
int IclaSize;
int i, j, MyNode=mynode();

  sprintf(FileName, "FILE%d", mynode());

  OCADp = PASSION_malloc_OCAD(DIMENSIONS, ROW_MAJOR);
  if (OCADp == (OCAD *)0)
  {
    printf("Error in OCAD.\n");
    exit(0);
  }
  if (PASSION_fill_OCAD(OCADp, global_size, distribution, nprocs, ocla_size,
                        icla_size,overlap, sizeof(double)) != 0)
  {
    printf("Error in filling OCAD.\n");
    exit(0);
  }

  if ((PFilePtr = PASSION_open(FileName, strlen(HEADER_STRING)+1)) == (PFILE *)0)
  {
    printf("Error in opening the file.\n");
    exit(0);
  }


  IclaSize = (icla_size[0][1] - icla_size[0][0] + 1) *
       (icla_size[1][1] - icla_size[1][0] + 1);
  Array = (double *)malloc(IclaSize * sizeof(double));

/* where does the section lie in the OCLA */

  AccessArray[0][0] = 0;
  AccessArray[0][1] = icla_size[0][1] - icla_size[0][0];
  AccessArray[0][2] = 2;
  AccessArray[1][0] = 0;
  AccessArray[1][1] = icla_size[1][1] - icla_size[1][0];
  AccessArray[1][2] = 2;

  count = 100.0;
    for (i=0; i<IclaSize; i++)
      Array[i] = count++;

  if (PASSION_write_section(PFilePtr, OCADp, (char *)Array, 0, 0,
        AccessArray) != 0)
  {
     printf("Error in PASSION_write_section.\n");
     exit(0);
  }

/* display.c can be used to view the file */

  if (PASSION_close(PFilePtr) == -1)
  {
    printf("Error in closing the file.\n");
    exit(0);
  }

  PASSION_free_OCAD(OCADp);
}
```

6.5 PROGRAM 5 : Using Prefetching

```
/* This is an example of prefetching using PASSION_read_prefetch */

#include "passion.h"

#define DIMENSIONS (unsigned int)2
#define HEADER_STRING "PASSION File Header"

void main(int argc, char **argv)
{
PFILE *PFilePtr;
int global_size[DIMENSIONS] = {16, 16};
int nprocs[DIMENSIONS] = {2, 1};
int distribution[DIMENSIONS][2] = {{BLOCK_DISTRIBUTION, -1},
                                   {NO_DISTRIBUTION, -1}
                                  };
int AccessArray[DIMENSIONS][3];
int overlap[DIMENSIONS][2] = {{0, 0}, {0, 0}};
int icla_size[DIMENSIONS][2] = {{0, 7}, {0, 15}};
int ocla_size[DIMENSIONS] = {8,16};
OCAD *OCADp;
double *Array;
char FileName[20];
int IclaSize;
int i, j, MyNode=mynode();
PREFETCH *PREFETCHp;

  sprintf(FileName, "FILE%d", mynode());

  OCADp = PASSION_malloc_OCAD(DIMENSIONS, ROW_MAJOR);
  if (OCADp == (OCAD *)0)
  {
    printf("Error in OCAD.\n");
    exit(0);
```

```
}

if (PASSION_fill_OCAD(OCADp, global_size, distribution, nprocs, ocla_size,
      icla_size, overlap,  sizeof(double)) != 0)
{
  printf("Error in filling OCAD.\n");
  exit(0);
}

if ((PFilePtr = PASSION_open(FileName,strlen(HEADER_STRING)+1)) == (PFILE *)0)
{
  printf("Error in opening the file.\n");
  exit(0);
}

IclaSize = (icla_size[0][1] - icla_size[0][0] + 1) *
   (icla_size[1][1] - icla_size[1][0] + 1);
Array = (double *)malloc(IclaSize * sizeof(double));

AccessArray[0][0] = 0;
AccessArray[0][1] = icla_size[0][1] - icla_size[0][0];
AccessArray[0][2] = 1;
AccessArray[1][0] = 0;
AccessArray[1][1] = icla_size[1][1] - icla_size[1][0];
AccessArray[1][2] = 1;

if ((PREFETCHp = PASSION_read_prefetch(PFilePtr, OCADp, (char *)Array,
      0, 0, AccessArray)) == 0)
{
   printf("Error in PASSION_read_prefetch.\n");
   exit(0);
 }

PASSION_prefetch_wait(PREFETCHp);

for (j=0; j<numnodes(); j++)
  {
    if (MyNode == j)
{
  printf("[%d]", MyNode);
  for (i = 0; i < IclaSize; i++)
    {
      printf("%.2f\t", (float)*(Array + i));
    }
  printf("\n");
}
    gsync();
  }

if (PASSION_close(PFilePtr) == -1)
{
    printf("Error in closing the file.\n");
    exit(0);
  }

  PASSION_free_OCAD(OCADp);
}
```

6.6 PROGRAM 6 : Using Data Reuse

```
/* This program illustrates read with reuse using PASSION_read_reuse */

#include <string.h>
#include "passion.h"

#define DIMENSIONS (unsigned int)2
#define HEADER_STRING "PASSION File Header"

void main(int argc, char **argv)
{
PFILE *PFilePtr;
int global_size[DIMENSIONS] = {16, 16};
int nprocs[DIMENSIONS] = {1, 1};
int distribution[DIMENSIONS][2] = {{NO_DISTRIBUTION, -1},
                                    {NO_DISTRIBUTION, -1}
                                   };
int overlap[DIMENSIONS][2] = {{1, 1}, {0, 0}};
int icla_size[DIMENSIONS][2] = {{0, 1}, {0, 15}};
int ocla_size[DIMENSIONS] = {10,16};
OCAD *OCADp;
REUSE *REUSEp;
double Array[4][16];
char FileName[20];
int IclaSize;
int i, j, MyNode=mynode(), retval;

  sprintf(FileName, "FILE%d", mynode());

  OCADp = PASSION_malloc_OCAD(DIMENSIONS, ROW_MAJOR);
  if (OCADp == (OCAD *)0)
  {
    printf("Error in OCAD.\n");
    exit(0);
  }
  if (PASSION_fill_OCAD(OCADp, global_size, distribution, nprocs, ocla_size,
          icla_size, overlap, sizeof(double)) != 0)
  {
    printf("Error in filling OCAD.\n");
    exit(0);
  }
  if ((PFilePtr = PASSION_open(FileName, strlen(HEADER_STRING)+1)) == (PFILE *)0)
  {
    printf("Error in opening the file.\n");
    return;
  }

  IclaSize = (icla_size[0][1] - icla_size[0][0] + 1) *
   (icla_size[1][1] - icla_size[1][0] + 1) + (overlap[0][0] +
            overlap[0][1])*(icla_size[1][1] - icla_size[1][0] + 1);

  REUSEp = PASSION_reuse_init(PFilePtr, OCADp, 0);
  if (REUSEp == (REUSE *)0)
    {
       printf("Error in reuse init.\n");
       exit(0);
    }

  while(!(retval = PASSION_read_reuse(REUSEp, (char *)Array)))
    {
       printf("[%d]", MyNode);
       for (i = 0; i <4; i++)
       {
   for (j = 0; j < 16; j++)
           printf("%.2f\t", (float)Array[i][j]);
         printf("\n");
       }
     }

  if (retval == -1) printf("Error in PASSION_read_reuse.\n");

  if (PASSION_close(PFilePtr) == -1)
  {
    printf("Error closing the file.\n");
    exit(0);
  }

  PASSION_free_OCAD(OCADp);
}
```

6.7 PROGRAM 7 : Reading Array Sections in the Global Placement Model

```
/* This program reads a section from the global array file using
   PASSION_global_read */

#include<stdio.h>
#include <fcntl.h>
#include <nx.h>
#include <math.h>
```

```
#include "passion.h"

#define DIMENSIONS (unsigned int)2
#define HEADER_STRING "PASSION File Header"

void main()
{
  PFILE *fp;
  int global_size[DIMENSIONS] = {8, 16};
  int nprocs[DIMENSIONS] = {2, 1};
  int distribution[DIMENSIONS][2] = {{BLOCK_DISTRIBUTION, -1},
                                     {NO_DISTRIBUTION, -1}
                                    };
  int AccessArray[DIMENSIONS][3];
  int overlap[DIMENSIONS][2] = {{0, 0}, {0, 0}};
  int icla_size[DIMENSIONS][2] = {{0, 7}, {0, 15}};
  int ocla_size[DIMENSIONS] = {8,16};
  double *Array;
  int mynod,file_block=4;
  int i,j,k;
  int l1,l2,u1,u2;
  int row,col;
  int NO_ROWS = 8, NO_COLS = 16;
  OCAD *OCADp;


  OCADp = PASSION_malloc_OCAD(DIMENSIONS, ROW_MAJOR);
  if (OCADp == (OCAD *)0)
  {
    printf("Error in OCAD.\n");
    exit(0);
  }
  if (PASSION_fill_OCAD(OCADp, global_size, distribution, nprocs, ocla_size,
         icla_size, overlap, sizeof(double)) != 0)
  {
    printf("Error in filling OCAD.\n");
    exit(0);
  }

  if ((fp = PASSION_open("FILE0", strlen(HEADER_STRING)+1)) == (PFILE *)0)
  {
    printf("Error in opening the file.\n");
    return;
  }

  Array = (double *)malloc(NO_ROWS*NO_COLS*sizeof(double));

  mynod = mynode();

/* section in global coordinates */
```

```
  AccessArray[0][0] = mynod*file_block;
  AccessArray[0][1] = mynod*file_block + 3;
  AccessArray[0][2] = 1;
  AccessArray[1][0] = 0;
  AccessArray[1][1] = 15;
  AccessArray[1][2] = 1;

  if (PASSION_global_read(fp, OCADp, (char *)Array, 0, 0, AccessArray,
        numnodes()) != 0)
  {
      printf("Error in PASSION_global_read.\n");
      exit(0);
  }


  l1 =  AccessArray[0][0];
  u1 =  AccessArray[0][1];
  l2 =  AccessArray[1][0];
  u2 =  AccessArray[1][1];

  for (k=0; k<numnodes(); k++) {
    if (mynode() == k) {
      for(i = 0,row = 0; i <= (u1-l1); i++ ,row++)
  {
    printf("MYNODE %d  ROW %d    ", mynod, row);
    for(j=0,col = 0; j <= (u2-l2); j ++,col++)
      {
        printf("%f\t ",*(Array + row*NO_COLS + col));
      }
    printf("\n");
  }
    }
    gsync();
  }

  if (PASSION_close(fp) == -1)
  {
    printf("Error in closing the file.\n");
    exit(0);
  }

  PASSION_free_OCAD(OCADp);
}
```

6.8 PROGRAM 8 : Writing Array Sections in the Global Placement Model

```
/* This program writes a section to the global array file using
   PASSION_global_write */

#include<stdio.h>
#include <fcntl.h>
#include <nx.h>
#include <math.h>
#include<malloc.h>
#include "passion.h"

#define DIMENSIONS (unsigned int)2
#define HEADER_STRING "PASSION File Header"

void main()
{
  PFILE *fp;
  int global_size[DIMENSIONS] = {8, 16};
  int nprocs[DIMENSIONS] = {2, 1};
  int distribution[DIMENSIONS][2] = {{BLOCK_DISTRIBUTION, -1},
                                     {NO_DISTRIBUTION, -1}
                                    };
  int AccessArray[DIMENSIONS][3];
  int overlap[DIMENSIONS][2] = {{0, 0}, {0, 0}};
  int icla_size[DIMENSIONS][2] = {{0, 7}, {0, 15}};
  int ocla_size[DIMENSIONS] = {8,16};
  double *Array, count=100.0;
  int mynod,file_block=4;
  int i,j,k;
  int l1,l2,u1,u2;
  int row,col;
  int NO_ROWS = 8, NO_COLS = 16;
  OCAD *OCADp;

  OCADp = PASSION_malloc_OCAD(DIMENSIONS, ROW_MAJOR);
  if (OCADp == (OCAD *)0)
  {
    printf("Error in OCAD.\n");
    exit(0);
  }
  if (PASSION_fill_OCAD(OCADp, global_size, distribution, nprocs, ocla_size,
         icla_size, overlap, sizeof(double)) != 0)
  {
    printf("Error in filling OCAD.\n");
    exit(0);
  }

  if ((fp = PASSION_open("FILE0", strlen(HEADER_STRING)+1)) == (PFILE *)0)  {
    printf("Error in opening the file.\n");
    return;
  }


  Array = (double *)malloc(NO_ROWS*NO_COLS*sizeof(double));
  mynod = mynode();

  AccessArray[0][0] = mynod*file_block;
  AccessArray[0][1] = mynod*file_block + 3;
  AccessArray[0][2] = 1;
  AccessArray[1][0] = 0;
  AccessArray[1][1] = 15;
  AccessArray[1][2] = 1;

  for(i=0; i<NO_ROWS; i++)
  {
    for(j=0; j<NO_COLS; j++)
    {
      *(Array + i*NO_COLS + j) = (count++) + mynod*200;
    }
  }

  if (PASSION_global_write(fp, OCADp, (char *)Array, 0, 0, AccessArray,
       numnodes()) != 0)
  {
      printf("Error in PASSION_global_write.\n");
      exit(0);
  }

  if (PASSION_close(fp) == -1)
  {
    printf("Error in closing the file.\n");
    exit(0);
  }

PASSION_free_OCAD(OCADp);
}
```

6.9 Sample Makefile

```
# Sample Makefile for PASSION Applications

# List the name of the directory where the PASSION include files
# is stored.
INCDIR=

# List the name of the directory where the PASSION runtime library
# is stored.
LIBDIR=
# Add the names of supporting libraries needed, e.g. Add -nx to
# when compiling on Intel PARAGON
LIBS= -lpassion -lm

create : create.o
  $(CC) -o create $(CCFLAGS) -L $(LIBDIR) create.o $(LIBS)

create.o : create.c
  $(CC) -c -D$(PASSION_ARCH) $(CCFLAGS) -I $(INCDIR) create.c
```

Chapter 7

Function Reference

C

Synopsis

int PASSION_close(PFilePtr);

Description

This routine is used to close the parallel file.

Parameter Declarations

PFILE *PFilePtr; *PFilePtr* is a file pointer returned by a previous *PASSION_open*.

Return Value

The function returns 0 if successful, else it returns -1[1].

F

Synopsis

int PASSION_fill_OCAD(OCADp, size, distribution, nprocs, ocla_size, icla_size, overlap, elemsize);

Description

This routine fills the *OCAD* for the array. *Only this routine should be used to fill the OCAD.*

Parameter Declarations

[1]All functions that return an integer use -1 as the return value to indicate error.

OCAD *OCADp; *OCADp* points to a previously allocated OCAD using *PASSION_malloc_OCAD. Please note that the sizes of the parameters to follow depend on the dimension specified in the OCAD. It is the user's responsibility to ensure that the dimension parameter used to create the OCAD structure is consistent with parameters given to the PASSION_fill_OCAD call to follow.*

int *size; *size* is a single dimensional array of integers. It contains the *global size* of the array in each dimension.

int distribution[][2]; *distribution* is a two dimensional array of integers. Row *i* of the array gives the distribution information about the array in dimension *i*. The first element in the row specifies the type of distribution. Valid values are :
NO_DISTRIBUTION
BLOCK_DISTRIBUTION
CYCLIC_DISTRIBUTION
The second element in the row gives the block size for the distribution. It is relevant only when the distribution is CYCLIC_DISTRIBUTION.

int *nprocs; *nprocs* is a one dimensional array of integers. It gives the number of processors over which the array is distributed in each dimension.

int *ocla_size; *ocla_size* is a one dimensional array of integers. It contains the size of the OCLA in each dimension.

int *icla_size[][2]; *icla_size* is a two dimensional array of integers. It contains the size of the ICLA in each dimension. The first element in the row specifies the lowest index of the ICLA in that dimension and the second element specifies the highest index of the ICLA in that dimension.

int overlap[][2]; *overlap* is a two dimensional array of integers. Row *i* of the array gives the overlap information about the array in dimension *i*. The first element in the row gives the size of lower overlap area in that dimension. The second element gives the size of upper overlap area in that dimension.

int elemsize; *elemsize* specifies the size of each array element in bytes.

Return Value

This function returns zero, if successful, else it returns -1.

Synopsis

void PASSION_free_OCAD(OCADp);

Description

This routine is used to deallocate the OCAD. *Only this routine should be used to deallocate the OCAD.*

Parameter Declarations

OCAD *OCADp; *OCADp* points to a previously allocated OCAD structure using *PASSION_malloc_OCAD.*

G

Synopsis

int PASSION_global_read(PFilePtr, OCADp, Array, i, j, AccessArray, nprocs);

Description

This is a collective I/O call which uses the Extended Two Phase Method to read an array section from a global file into main memory. The same file should be specified for all processors. The section to be read is specified in AccessArray. Data is written into Array starting from location (i,j). Data is stored without stride in the Array, even if there is a stride specified in the out-of-core global array. This routine is for the Global Placement Model only.

Parameter Declarations

PFILE *PFilePtr; *PFilePtr* is the file pointer of the file to be read.

OCAD *OCADp; *OCADp* is a pointer to the OCAD of the out-of-core array.

char *Array; *Array* is the pointer to the start of the buffer into which data is to be read.

int i, j; These parameters specify the location in Array from where the section is to be stored.

int AccessArray[][3]; *AccessArray* specifies the section to be read in global coordinates. See Section 3.5 for details about how to specify a section.

int nprocs; This gives the number of processors reading the data. nprocs should always be equal to the number of processors on which the program is being executed.

Return Value

This function returns 0 if successful, else it returns -1.

Synopsis

int PASSION_global_write(PFilePtr, OCADp, Array, i, j, AccessArray, nprocs);

Description

This is a collective I/O call which uses the Extended Two Phase Method to write an array section to a global file. The same file should be specified for all processors. The section to be written is specified in AccessArray. Data is written starting from location (i,j) in Array. Data is assumed to be stored without stride in Array, but is written with the specified stride in the global array file. This routine is for the Global Placement Model only.

Parameter Declarations

PFILE *PFilePtr; *PFilePtr* is the file pointer of the file into which data is to be written.

OCAD *OCADp; *OCADp* is the pointer to the OCAD of the out-of-core array.

char *Array; *Array* is the pointer to the start of the buffer from which data is to be written.

int i, j; These parameters specify the location in Array from where data is to be written to the file.

int AccessArray[][3]; *AccessArray* specifies the section to be written in global coordinates. See Section 3.5 for details about how to specify a section.

int nprocs; This gives the number of processors writing data. nprocs should always be equal to the number of processors on which the program is being executed.

Return Value

This function returns 0 if successful, else it returns -1.

M

Synopsis

OCAD *PASSION_malloc_OCAD(dimensions, storage);

Description

This routine is used to allocate the OCAD. *Only this routine should be used to allocate the OCAD.*

Parameter Declarations

int dimensions; *dimensions* is the number of dimensions of the array.

int storage; *storage* specifies the way the array is stored in the file. It can be either *ROW_MAJOR* or *COLUMN_MAJOR. Currently only ROW_MAJOR storage is supported.*

Return Value

The function returns a pointer to the OCAD structure (see Section 3.1) if successful, else it returns a NULL pointer (*(OCAD *)0*).

O

Synopsis

PFILE *PASSION_open(filename, headersize);

Description

This routine is used to open the parallel file. *Only this routine should be used to open the file.*

Parameter Declarations

char *filename; *filename* points to a character string containing the name of the file to be opened.

int headersize; *headersize* specifies the size of the header at the start of the file, in bytes.

Return Value

The function returns a *file pointer* (see Section 3.2) if successful, else it returns *NULL* pointer((PFILE *)0).

P

Synopsis

int PASSION_prefetch_wait(PREFETCHp);

Description

This routine is used to wait for a previous prefetch read to complete. (See Section 5.3.2 for more details about prefetching.) This routine is for the Local Placement Model only.

Parameter Declarations

Parameters :

PREFETCH *PREFETCHp; *PREFETCHp* is a pointer to the corresponding PREFETCH structure returned by a previous *PASSION_read_preftech*.

Return Value

The routine returns 0 if successful and -1 otherwise.

R

Synopsis

int PASSION_read(PFilePtr, OCADp, Array);

Description

This routine is used to read the entire OCLA into Array. Hence the size of Array should at least be equal to that of the OCLA. This routine is for the Local Placement Model only.

Parameter Declarations

PFILE *PFilePtr; *PFilePtr* is the file pointer of the file to be read.

OCAD *OCADp; *OCADp* is a pointer to the OCAD of the out-of-core array.

char *Array; *Array* is a pointer to the start of the ICLA.

Return Value

The function returns 0 if successful, else it returns −1.

Synopsis

int PASSION_read_header(PFILE *PFilePtr, char *HBuf);

Description

This routine is used to read the header at the start of the array file into a buffer.

Parameter Declarations

PFILE *PFilePtr; *PFilePtr* is the file pointer of the file to be read.

char *Hbuf; *Hbuf* is a pointer to the start of the buffer.

Return Value

The function returns 0 if successful, else it returns −1.

Synopsis

PREFETCH *PASSION_read_prefetch(PFilePtr, OCADp, Array, i, j, AccessArray);

Description

This routine is used for prefetching an array section. (See Section 5.3.2 for more details about prefetching). It initiates one or more read operations and returns control back to the user program. This routine is for the Local Placement Model only.

Parameter Declarations

PFILE *PFilePtr; *PFilePtr* is the file pointer of the file to be read.

OCAD *OCADp; *OCADp* is a pointer to the OCAD of the array.

char *Array; *Array* is a pointer to the start of the ICLA.

int i, j; These parameters indicate the location in Array from where the section is to be stored.

int AccessArray[][3]; *AccessArray* specifies the section to be read in local coordinates. See Section 3.5 for details about how to specify an array section.

Return Value

The routine returns a pointer to the PREFETCH structure if successful, else it returns *(PREFETCH *)0*. The PREFETCH structure pointer (See Section 3.3) should be used as parameter to the function *PASSION_prefetch_wait* to wait for the read to finish.

Synopsis

int PASSION_read_section(PFilePtr, OCADp, Array, i, j, AccessArray);

Description

This routine is used to read array sections from the local array file using Data Sieving. The section is stored without stride in Array, even if there is a stride in the OCLA. The shape of the section is retained in the ICLA. This routine is for the Local Placement Model only.

Parameter Declarations

PFILE *PFilePtr; *PFilePtr* is the file pointer of the file to be read.

OCAD *OCADp; *OCADp* is the pointer to the OCAD of the array.

char *Array; *Array* is a pointer to the beginning of the ICLA.

int i, j; These parameters indicate the location in Array from where the section is to be read.

int AccessArray[][3]; *AccessArray* specifies the section to be read in local coordinates. See Section 3.5 for details about how to specify an array section.

Return Value

The function returns 0 if successful, else it returns -1.

Synopsis

REUSE *PASSION_reuse_init(PFilePtr, OCADp, StartRow);

Description

This routine initiates a read with reuse operation. (See Section 5.3.3 for more details about data reuse.) *This routine has to be called before calling the PASSION_read_reuse routine.* This routine is for the Local Placement Model only.

Parameter Declarations

PFILE *PFilePtr; *PFilePtr* is the file pointer of the file to be read.

OCAD *OCADp; *OCADp* is the pointer to the OCAD of the array.

int StartRow; *StartRow* is the starting row from where the reuse operation will start.

Return Value

The routine returns a pointer to the REUSE structure (see Section 3.4) if successful. Else it returns *(REUSE *)0.*

Synopsis

int PASSION_read_reuse(REUSEp, Array);

Description

This routine reads the next block of data from the array with reuse. (See Section 5.3.3 for more details about data reuse.) This routine is for the Local Placement Model only.

Parameter Declarations

REUSE *REUSEp; *REUSEp* is a pointer to the REUSE structure (see Section 3.4) returned by an earlier *PASSION_reuse_init* call.

char *Array; *Array* is a pointer to the ICLA in which the data is to be read.

Return Value

The routine returns 0 if successful. It returns 1 if it has reached the end of the array. It returns -1 if any error occurs.

W

Synopsis

int PASSION_write(PFilePtr, OCADp, Array);

Description

This routine is used to write the entire ICLA to the OCLA. Hence the size of Array should at least be equal to that of the OCLA. This routine is for the Local Placement Model only.

Parameter Declarations

PFILE *PFilePtr; *PFilePtr* is the file pointer of the file to be written.

OCAD *OCADp; *OCADp* is the pointer to the OCAD of the array.

char *Array; *Array* is a pointer to the beginning of the ICLA.

Return Value

The function returns 0 if successful, else it returns -1.

Synopsis

int PASSION_write_header(PFILE *PFilePtr, char *HBuf);

Description

This routine is used to write the header from Hbuf to the start of the array file.

Parameter Declarations

PFILE *PFilePtr; *PFilePtr* is the file pointer of the file to be read.

char *Hbuf; *Hbuf* is a pointer to the start of the buffer containing the header.

Return Value

The function returns 0 if successful, else it returns −1.

Synopsis

int PASSION_write_section(PFilePtr, OCADp, Array, i, j, AccessArray);

Description

This routine is used to write array sections to the file using Data Sieving. The section is assumed to be without stride in Array, and is written with the specified stride to the OCLA. This routine is for the Local Placement Model only.

Parameter Declarations

PFILE *PFilePtr; *PFilePtr* is the file pointer of the file to be written.

OCAD *OCADp; *OCADp* is the pointer to the OCAD of the array.

char *Array; *Array* is a pointer to the beginning of the ICLA.

int i, j; These parameters indicate the location in Array from where the section is to be written to the file.

int AccessArray[][3]; *AccessArray* specifies the section to be written in local coordinates. See Section 3.5 for details about how to specify an array section.

Return Value

The function returns 0 if successful, else it returns −1.

Chapter 8

Installing the Software

8.1 How to download the software

The PASSION Runtime Library is available from the anonymous FTP sites :

- `erc.cat.syr.edu: ece/choudhary/PASSION/version-1.0`,
- `ftp.npac.syr.edu: users/choudhar/PASSION/version-1.0`.

The software is stored as a compressed tar file `passion.tar.Z`. The PASSION Runtime Library is also available on the World Wide Web at
`http://www.cat.syr.edu/passion.html`

8.2 How to unpack the software

1. Uncompress the downloaded file as follows :
   `%uncompress passion.tar.Z`
2. Untar the file `passion.tar` as follows :
   `%tar xf passion.tar`
3. This will create a passion-1.0 subdirectory in the current directory.

8.3 How to compile the software

There is a makefile provided in the PASSION directory. Edit the makefile and provide the necessary information. Then use the command:
% make

Chapter 9

Where to look for help

9.1 Comments and Feedback

For any questions, comments and bug reports regarding the PASSION Runtime Library, please contact the PASSION group by sending email to `passion@cat.syr.edu`.

9.2 Related Publications

Additional information about PASSION is available on the World Wide Web at
`http://www.cat.syr.edu/passion.html`.
PASSION related papers can also be obtained from the anonymous ftp sites :

- `erc.cat.syr.edu: ece/choudhary/PASSION`,
- `ftp.npac.syr.edu: users/choudhar/PASSION`.

The following is the list of papers related to the PASSION project and their corresponding file names:-

- `passion_report.ps.Z`: "PASSION: Parallel and Scalable Software for Input-Output", Alok Choudhary, Rajesh Bordawekar, Michael Harry, Rakesh Krishnaiyer, Ravi Ponnusamy, Tarvinder Singh and Rajeev Thakur, *NPAC Technical Report SCCS–636*, Sept. 1994.
- `extended-two-phase.ps.Z`: "Accessing Sections of Out-of-Core Arrays Using an Extended Two-Phase Method", Rajeev Thakur and Alok Choudhary, *NPAC Technical Report SCCS–685*, Jan. 1995.
- `ics94-out-of-core-hpf.ps.Z`: "Compiler and Runtime Support for Out-of-Core HPF Programs", Rajeev Thakur, Rajesh Bordawekar and Alok Choudhary, *Proc. of Int. Conf. on Supercomputing (ICS 94)*, July 1994, pp. 382-391.
- `splc94_passion_runtime.ps.Z`: "PASSION Runtime Library for Parallel I/O", Rajeev Thakur, Rajesh Bordawekar, Alok Choudhary, Ravi Ponnusamy and Tarvinder Singh, *Proc. of the Scalable Parallel Libraries Conference*, Oct. 1994
- `access_reorg.ps.Z`: "Data Access Reorganizations in Compiling Out-of-core Data Parallel Programs on Distributed Memory Machines ", Rajesh Bordawekar, Alok Choudhary and Rajeev Thakur, *NPAC Technical Report SCCS–622*, Sept. 1994.
- `vipfs.ps.Z`: "The Design of VIP-FS: A Virtual Parallel File System for High Performance Parallel and Distributed Computing", Juan Miguel del Rosario, Michael Harry and Alok Choudhary, *NPAC Technical Report SCCS–628*, May 1994.

- `adopt.ps.Z`: "ADOPT: A Dynamic Scheme for Optimal Prefetching in Parallel File Systems", Tarvinder Singh and Alok Choudhary, *NPAC Technical Report SCCS–627*, 1994.
- `task_data.ps.Z`: "Integrating Task and Data Parallelism Using Parallel I/O Techniques", Bhaven Avalani, Alok Choudhary, Ian Foster and Rakesh Krishnaiyer, *Proc. of the Int. Workshop on Parallel Processing*, Bangalore, India, Dec. 1994.

Bibliography

[BC94] R. Bordawekar and A. Choudhary. Language and Compiler Support for Parallel I/O. In *Proceedings of IFIP Working Conference on Programming Environments for Massively Parallel Distributed Systems*, April 1994.

[BdRC93] R. Bordawekar, J. del Rosario, and A. Choudhary. Design and Evaluation of Primitives for Parallel I/O. In *Proceedings of Supercomputing '93*, pages 452–461, November 1993.

[CBH$^+$94] A. Choudhary, R. Bordawekar, M. Harry, R. Krishnaiyer, R. Ponnusamy, T. Singh, and R. Thakur. PASSION: Parallel and Scalable Software for Input-Output. Technical Report SCCS–636, NPAC, Syracuse University, September 1994. Also available as CRPC Technical Report CRPC–TR94483–S.

[CFH$^+$94] P. Corbett, D. Feitelson, Y. Hsu, J. Prost, M. Snir, S. Fineberg, B. Nitzberg, B. Traversat, and P. Wong. MPI-IO: A Parallel I/O Interface for MPI, Version 0.2. Technical Report IBM Research Report RC 19841(87784), IBM T. J. Watson Research Center, November 1994.

[dRBC93] J. del Rosario, R. Bordawekar, and A. Choudhary. Improved Parallel I/O via a Two-Phase Runtime Access Strategy. In *Proceedings of the Workshop on I/O in Parallel Computer Systems at IPPS '93*, April 1993.

[dRHC94] J. del Rosario, M. Harry, and A. Choudhary. The Design of VIP-FS: A Virtual Parallel File System for High Performance Parallel and Distributed Computing. Technical Report SCCS-628, NPAC, Syracuse University, May 1994.

[SC94] T. Singh and A. Choudhary. ADOPT: A Dynamic Scheme for Optimal Prefetching in Parallel File Systems. Technical Report SCCS–627, NPAC, Syracuse University, 1994.

[TBC94a] R. Thakur, R. Bordawekar, and A. Choudhary. Compiler and Runtime Support for Out-of-Core HPF Programs. In *Proceedings of the 8^{th} ACM International Conference on Supercomputing*, pages 382–391, July 1994.

[TBC$^+$94b] R. Thakur, R. Bordawekar, A. Choudhary, R. Ponnusamy, and T. Singh. PASSION Runtime Library for Parallel I/O. In *Proceedings of the Scalable Parallel Libraries Conference*, October 1994.

[TC95] R. Thakur and A. Choudhary. Accessing Sections of Out-of-Core Arrays Using an Extended Two-Phase Method. Technical Report SCCS–685, NPAC, Syracuse University, January 1995. Also available as CRPC Technical Report CRPC–TR95508–S.

Index