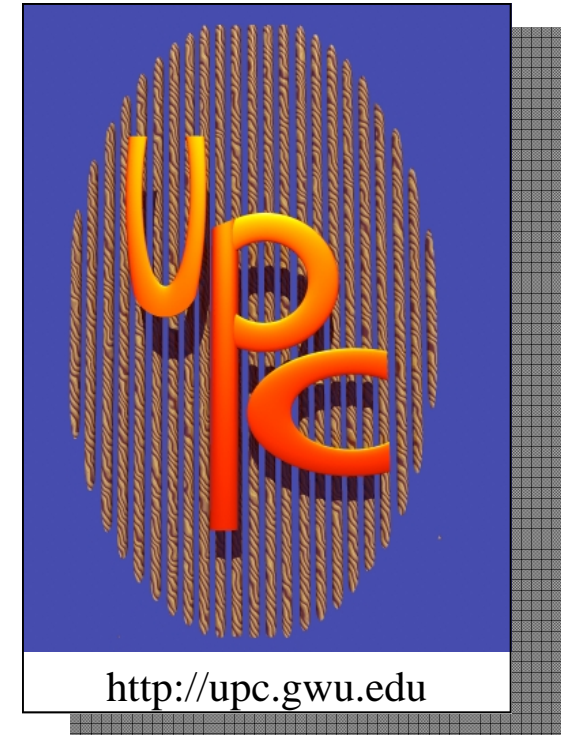
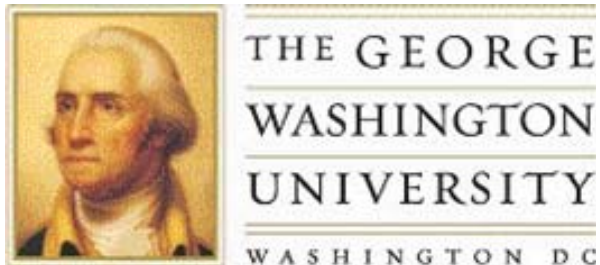


UPC-IO



A library interface for parallel I/O in UPC

An Overview - *Based on UPC-IO Specs v1.0*



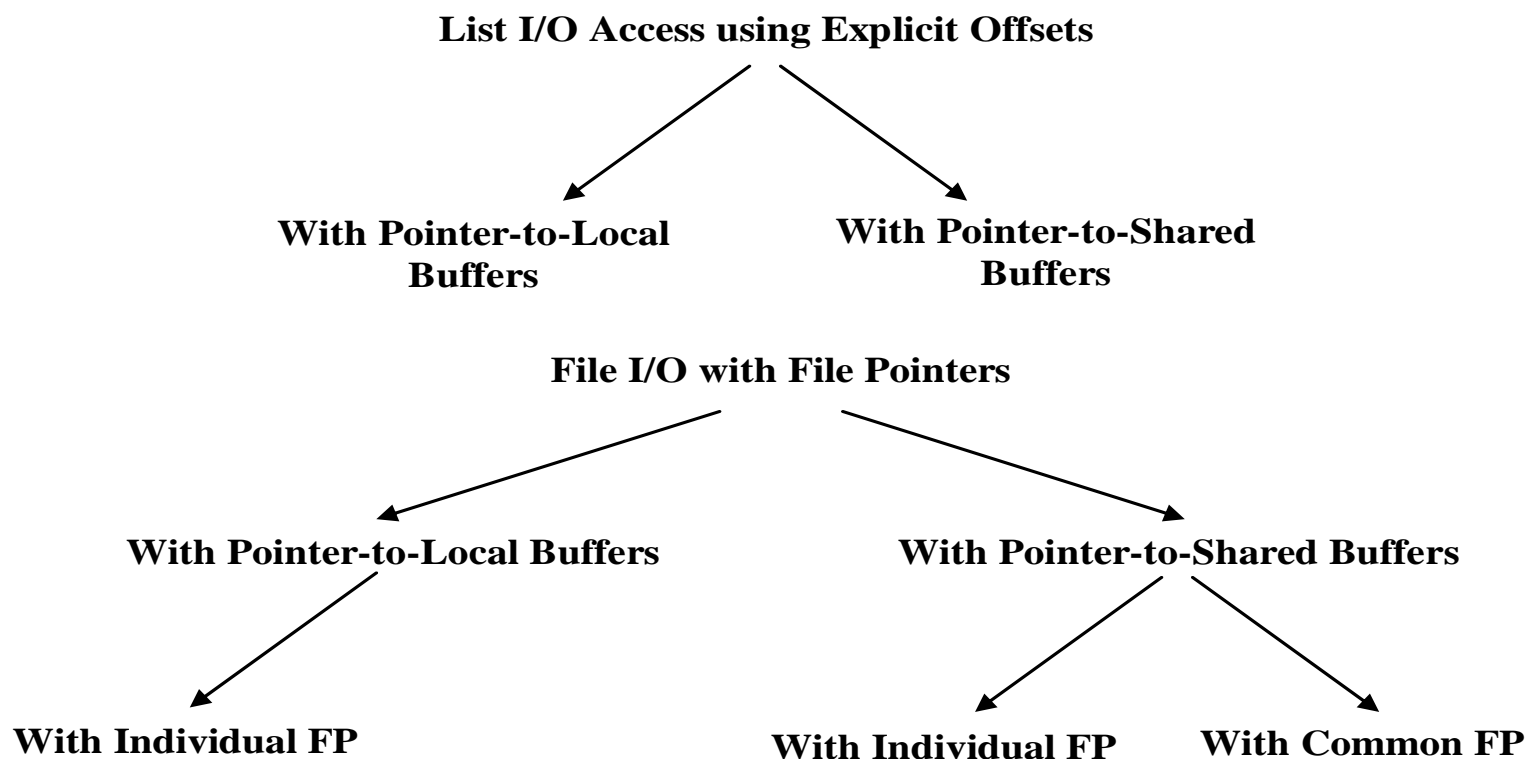
- ◆ **Brief overview of UPC-IO**
- ◆ **Major functions defined by UPC-IO**
- ◆ **UPC-IO Examples**
- ◆ **Recent changes in UPC-IO Spec (pre10 to v1.0)**

Brief overview of UPC-IO

- ◆ All UPC-IO functions are collective
 - Most arguments are single-valued
- ◆ Library based API
 - Enables ease of plugging into existing parallel IO facilities (eg MPI-IO)
- ◆ UPC IO data movement operations support:
 - shared or private buffers
 - Two kinds of file pointers:
 - ◆ individual (ie per-thread)
 - ◆ common (ie shared by all threads)
 - Blocking and non-blocking (async) operations
 - ◆ Currently limited to one async op in-flight per file handle

- ◆ **UPC-IO also supports List-IO Access**
 - **Each thread provides a list of:**
 - ◆ (address, length) tuples referencing shared or private memory
 - ◆ (file offset, length) tuples referencing the file
 - ◆ Supports arbitrary size combinations, with very few semantic restrictions (eg no overlap within a list)
 - **Works entirely independently of file pointers**
 - ◆ Does not use or modify any file pointers
 - **Fully general I/O operation**
 - ◆ General enough to implement all the other data movement functions, although less convenient for common usage

File Access Methods in UPC-IO



all read/write operations have blocking and asynchronous (non-blocking) variants

Consistency & Atomicity Semantics of File Data

- ◆ UPC-IO consistency semantics define when the file data written by a thread is visible to other threads via file reads
- ◆ UPC-IO atomicity semantics define the outcome of overlapping concurrent write operations to the file from separate threads
 - reads and writes from a *single* thread (using individual file pointer) which don't conflict with other threads are always guaranteed to see consistent results (we preserve same-thread dependencies)
- ◆ Default mode: weak semantics (high performance)
 - overlapping or conflicting accesses have undefined results, unless they are separated by a `upc_all_fsync` (or a file close)
- ◆ `UPC_STRONG_CA` mode: strong semantics (low performance)
 - provides well-defined, deterministic semantics for overlapping or conflicting accesses (see spec for details)
- ◆ The semantic mode is chosen at file open time, and can be changed or queried for an open file handle using `upc_all_fcntl`

- ◆ **upc_all_fopen**
- ◆ **upc_all_fclose**
- ◆ **upc_all_fseek**
- ◆ **upc_all_fsync**
- ◆ **upc_all_fset_size**
- ◆ **upc_all_fget_size**
- ◆ **upc_all_fpreallocate**
- ◆ **upc_all_fcntl**
- ◆ **upc_all_fread_local [_async]**
- ◆ **upc_all_fread_shared [_async]**
- ◆ **upc_all_fwrite_local [_async]**
- ◆ **upc_all_fwrite_shared [_async]**
- ◆ **upc_all_fread_list_local [_async]**
- ◆ **upc_all_fread_list_shared [_async]**
- ◆ **upc_all_fwrite_list_local [_async]**
- ◆ **upc_all_fwrite_list_shared [_async]**
- ◆ **upc_all_fwait_async**
- ◆ **upc_all_ftest_async**

Major UPC-IO functions

```
upc_file_t *upc_all_fopen( const char *filename, int flags,  
                           size_t numhints, upc_hints_t const *hints)
```

- ◆ Opens file, returns a pointer to a file descriptor, or NULL w/ errno
- ◆ File open flags:
 - Must explicitly choose one read/write mode:
 - ◆ **UPC_RDONLY** or **UPC_WRONLY** or **UPC_RDWR**
 - and one file pointer mode:
 - ◆ **UPC_INDIVIDUAL_FP** or **UPC_COMMON_FP**
 - other optional flags (analogous to POSIX):
 - ◆ **UPC_APPEND**, **UPC_CREATE**, **UPC_EXCL**, **UPC_TRUNC**
 - ◆ **UPC_DELETE_ON_CLOSE**
 - UPC-specific optional flags:
 - ◆ **UPC_STRONG_CA** - select strong consistency/atomicity mode
- ◆ Optional list of hints: for providing parallel I/O access hints

upc_off_t **upc_all_fseek**(upc_file_t *fd, upc_off_t offset, int origin)

- ◆ sets and queries the current position of the file pointer for fd
- ◆ Threads pass an origin analogous to C99:
 - UPC_SEEK_SET or UPC_SEEK_CUR or UPC_SEEK_END
 - and an offset from that origin
- ◆ All arguments must be single-valued for common file pointer
 - may be thread-specific for individual file pointer
- ◆ Returns the new file pointer position to each thread
 - Can also be used to simply query file pointer position using:
upc_all_fseek(fd, 0, UPC_SEEK_CUR)
- ◆ Seeking past EOF is permitted
 - writes past EOF will grow the file (filling with undefined data)

```
ssize_t upc_all_fread_local( upc_file_t *fd,  
                             void *buffer,  
                             size_t size, size_t nmemb,  
                             upc_flag_t sync_mode)
```

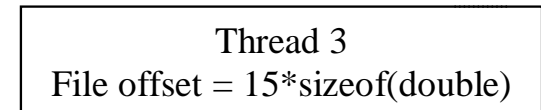
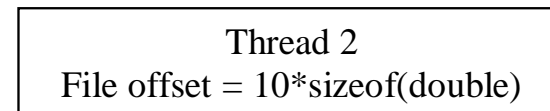
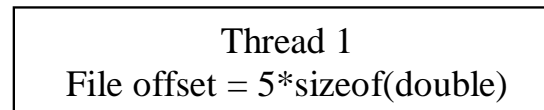
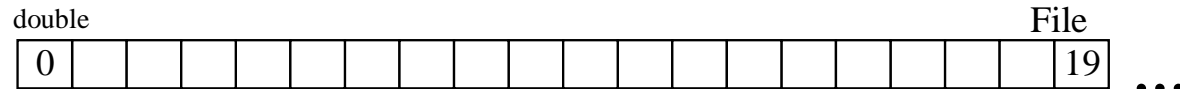
- ◆ Reads data from a file into a private buffer on each thread
 - File must be open for reading with individual file pointers
 - Each thread passes independent buffers and sizes
 - Number of bytes requested is $\text{size} * \text{nmemb}$ (*may be zero for no-op*)
- ◆ Returns the number of bytes successfully read for each thread
 - Each individual file pointer is incremented by corresponding amount
 - May return less than requested to indicate EOF
 - On error, it returns -1 and sets `errno` appropriately
- ◆ `upc_all_fwrite_local` takes same arguments & works analogously

```
ssize_t upc_all_fread_shared( upc_file_t *fd,  
                             shared void *buffer, size_t blocksize,  
                             size_t size, size_t nmemb,  
                             upc_flag_t sync_mode)
```

- ◆ **Reads data from a file into a shared buffer in memory**
 - Number of bytes requested is $size * nmemb$ (*may be zero for no-op*)
 - Buffer may be an arbitrarily blocked array, but input phase ignored (assumed 0)
- ◆ **When file is open for reading with an *individual* file pointer:**
 - Each thread passes independent buffers and sizes
 - Returns the number of bytes successfully read for each thread and increments individual file pointers by corresponding amount
- ◆ **When file is open for reading with a *common* file pointer:**
 - All threads pass same buffer and sizes (all arguments single-valued)
 - Returns the total number of bytes successfully read to all threads and increments the common file pointer by a corresponding amount
- ◆ **upc_all_fwrite_shared takes same arguments & works analogously**

UPC-IO Examples

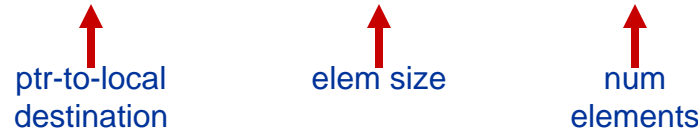
ARGONNE NATIONAL LABORATORY



```
double buffer[10]; // and assuming a
... // total of 4 THREADS
upc_file_t *fd =
    upc_all_fopen( "file", UPC_RDONLY | UPC_INDIVIDUAL_FP,
                  0, NULL ); /* no hints */

upc_all_fseek( fd, 5*MYTHREAD*sizeof(double), UPC_SEEK_SET );

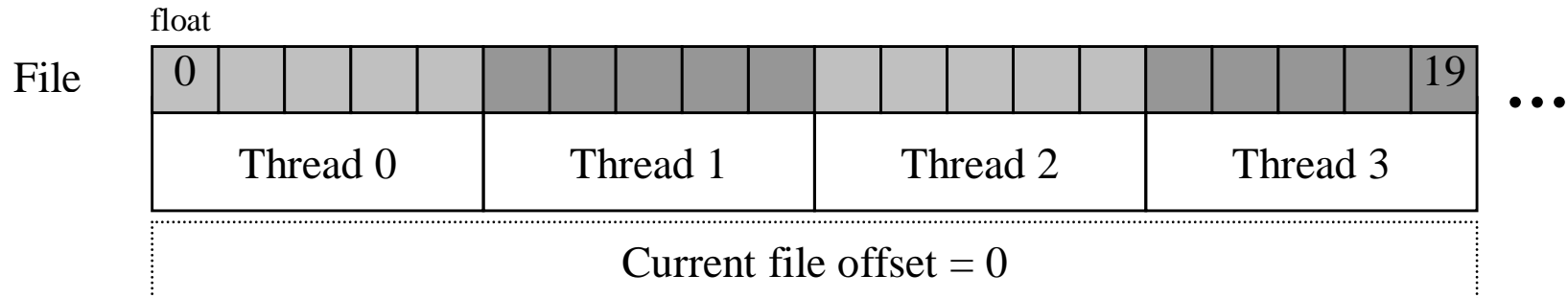
upc_all_fread_local( fd, buffer, sizeof( double ), 10, 0 );
upc_all_fclose(fd);
```



With individual file pointers, each thread operates independently and may pass diff. arguments (although they all still must make the collective call)

Example 1: Collective read into private can provide canonical file-view

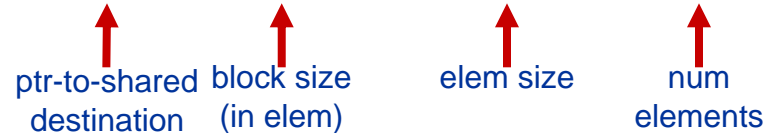
UPC-IO Examples (cont.)



```
shared [5] float buffer[20]; // and assuming a total of 4 static THREADS
```

```
...
upc_file_t *fd =
    upc_all_fopen( "file", UPC_RDONLY | UPC_COMMON_FP, 0, NULL );
```

```
upc_all_fread_shared( fd, buffer, 5, sizeof(float), 20, 0 );
```



 ptr-to-shared destination block size (in elem) elem size num elements

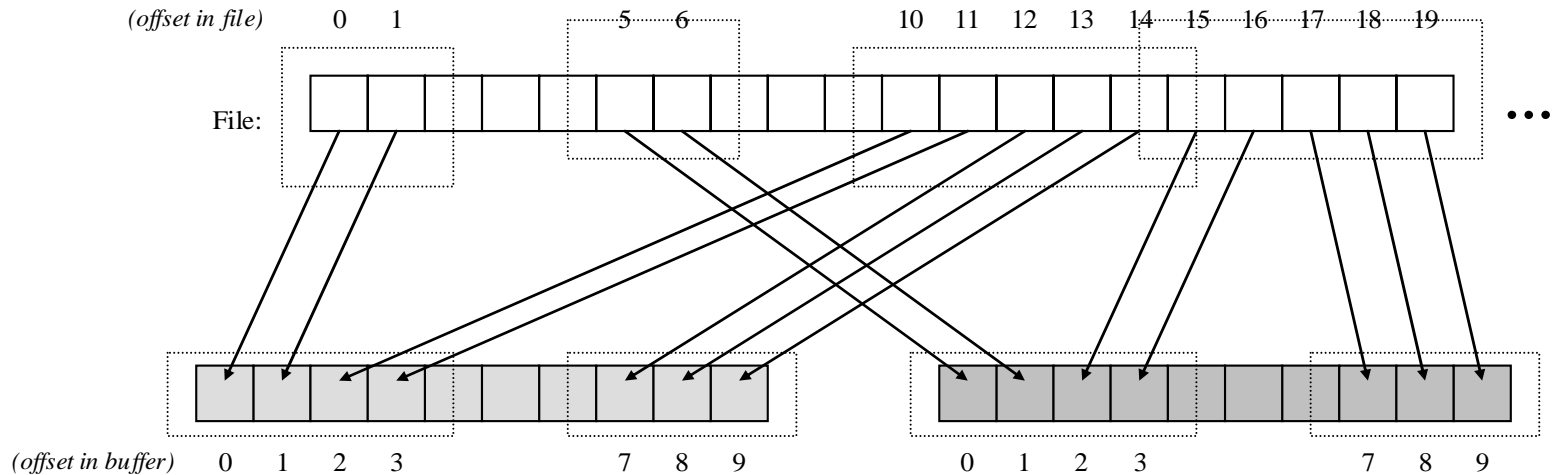
With common file pointers, all thread must pass the same (single-valued) arguments

sync mode
 0 equivalent to:

UPC_IN_ALLSYNC | UPC_OUT_ALLSYNC

Example 2: Read into a blocked shared buffer can provide a partitioned file-view

UPC-IO Examples (cont.)



```

char buffer[12];
upc_local_memvec_t memvec[2] = { { &buffer[0], 4 }, { &buffer[7], 3 } };

upc_filevec_t filevec[2] = { { MYTHREAD*5, 2 }, { 10+MYTHREAD*5, 5 } };

upc_file_t *fd =
    upc_all_open( "file", UPC_RDONLY | UPC_INDIVIDUAL_FP, 0, NULL);
upc_all_fread_list_local( fd, 2, &memvec, 2, &filevec,
    UPC_IN_ALLSYNC|UPC_OUT_ALLSYNC);

```

File pointer is irrelevant for list I/O, but one must be selected to open the file

Each thread passes an independent list of memory / file vectors

Example 3: I/O read of noncontiguous parts of a file to private noncontiguous buffers

Recent changes in UPC-IO Spec (pre10 to v1.0)

- ◆ **Changes to make UPC-IO less like POSIX and more like ISO C99**
 - Function prefix name changes, all now have prefix "upc_all_f"
 - ◆ eg. upc_all_open → upc_all_fopen, upc_all_file_whatever → upc_all_fwhatever
 - Separate size argument of data movement functions into numelements and elementsize, shared blocksize now passed in units of elements
 - Specified errno and error behavior for all functions
- ◆ **Changes to conform with new UPC Spec 1.2 terminology**
 - Functions taking pointer-to-local now named w/ _local suffix
 - ◆ e.g. upc_all_fread_private → upc_all_fread_local
- ◆ **Other miscellaneous naming changes for uniformity & clarity**
 - ◆ Private file pointer → individual file pointer
 - ◆ Shared file pointer → common file pointer
 - ◆ UPC_STRICT → UPC_STRONG_CA
 - ◆ _async suffix moved to end of all relevant function names

Recent changes in UPC-IO Spec (pre10 to v1.0)

- ◆ **Changes to conform with UPC Collectives Spec 1.0**
 - **Add `upc_flag_t sync_mode` argument to all data movement operations**
 - ◆ including the `_local` functions, to handle pointer-to-local aliasing shared data
 - **Same `upc_flag_t` values, operating analogously to UPC collectives spec**
 - ◆ `UPC_{IN,OUT}_{ALL,MY,NO}SYNC` - with same semantics and defaulting for 0
 - ◆ Governs semantics of data accesses to user-provided memory buffers (only)
 - ◆ Does not affect consistency of on-disk data (governed by `UPC_STRONG_CA` mode), or the behavior of internal library buffers
 - **Also extended the semantics of these flags to be meaningful for split-phase (async) UPC-IO operations**
 - ◆ Both flags are provided at async function initiation time
 - ◆ `UPC_IN_XSYNC` flag governs the synchronization at entry to the async initiation function
 - ◆ `UPC_OUT_YSYNC` flag governs the synchronization at exit from the async completion function (`upc_fwait_async` or successful `upc_ftest_async`)
 - ◆ Same semantics should probably be adopted when the collectives spec is extended to support explicitly non-blocking collective operations

- ◆ **Possible extensions to consider for a future library version**
 - Based on implementation/usage experience and user demand
- ◆ **Allow multiple asynchronous operations in-flight per file handle**
 - can be added later as a backwards-compatible extension, using explicit handles for each operation
 - `fread/fwrite_list_async` already provide sufficient generality for expressing most async I/O patterns we envision
- ◆ **Support changing read/write file mode for an open file handle using `upc_all_fcntl`**
 - can trivially be implemented using `close` and `reopen`, if somebody expresses a good motivation for wanting it
- ◆ **Whatever else we discover from implementation/usage experience**
 - Need to officially adopt spec so we can start implementing!!!