

Portable Parallel I/O

SIONlib

March 28, 2012 | Wolfgang Frings, Florian Janetzko, Michael Stephan

Outline

Introduction

- Motivation

- SIONlib in a NutShell

- SIONlib file format

- Details

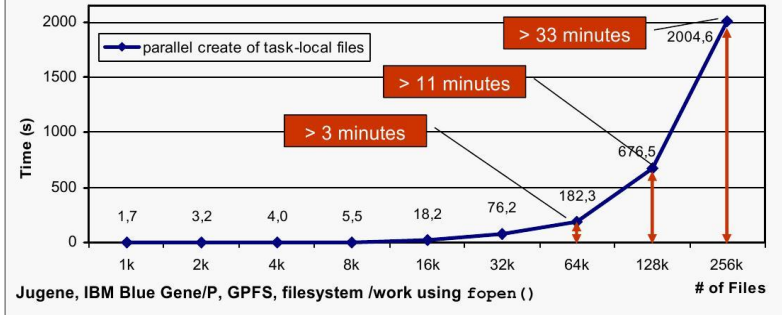
Interface

Example

Tools

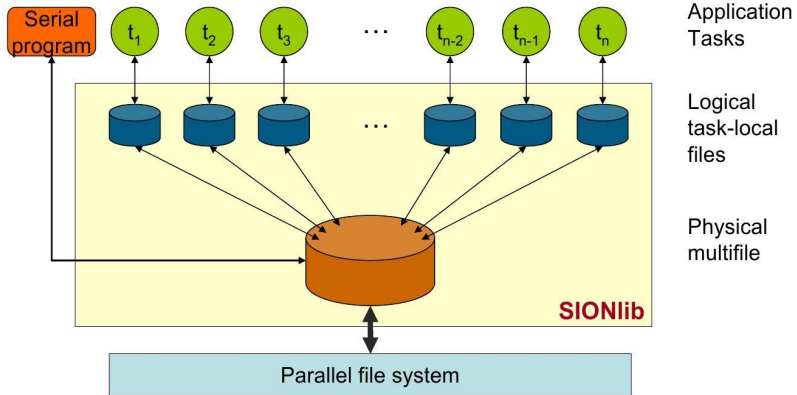
Motivation: Limitations of Task-Local I/O

Example: Creating files in parallel in the same directory



- Contention at the meta data server
- May degrade system I/O performance also for other users
- complicated file handling (e.g. archive)

Motivation: Using Shared Files



- Idea: Mapping many logical files onto one or a few physical file(s)
- → Task-local view to local data not changed

Introduction to SIONlib

- **SIONlib**: Scalable I/O library for parallel access to task-local files
- Collective I/O to binary shared files
- Logical task-local view to data
- Write and Read of binary stream-data
- Meta-Data Header/Footer included in file
- Collective open/close, independent write/read
- Write/read: POSIX or ANSI-C calls
- Support for MPI, OpenMP, MPI+OpenMP
- C, C++, and Fortran-wrapper
- Optimized for large processor numbers
(e.g. 288k tasks on Blue Gene/P Jugene)

Parallel I/O for Large Scale Application, Types

- External Formats:
 - Exchange data with others → portability
 - Pre- and Post-Processing on other systems (workflow)
 - Store data without system-dependent structure (e.g. number of tasks)
 - Archive data (long-term readable and self-describing formats)
- Internal Formats:
 - Scratch files, Restart files
 - Fastest I/O preferred
 - Portability and flexibility criteria of second order
 - Write and read data “as-is” (memory dump)
- SIONlib could support I/O of internal formats

SIONlib in a NutShell: Task local I/O

```
/* Open */  
sprintf(tmpfn, "%s.%06d",filename,my_nr);  
fileptr=fopen(tmpfn, "bw", ...);  
...  
/* Write */  
fwrite(bindata,1,nbytes,fileptr);  
...  
/* Close */  
fclose(fileptr);
```

- Original ANSI C version
- no collective operation, no shared files
- data: stream of bytes

SIONlib in a NutShell: Add SIONlib

```
/* Collective Open */
nfiles=1; chunksize=nbytes;
sid=sion_paropen_mpi( filename, "bw", &nfiles, &chunksize,
                    MPI_COMM_WORLD, &lcomm, &fileptr, ...);
...
/* Write */
fwrite(bindata,1,nbytes,fileptr);
...
/* Collective Close */
sion_parclose_mpi(sid);
```

- Collective (SIONlib) open and close
- Ready to run ...
- Parallel I/O to one shared file

SIONlib in a NutShell: Variable Data Size

```
/* Collective Open */
nfiles=1; chunksize=nbytes;
sid=sion_paropen_mpi( filename, "bw", &nfiles, &chunksize,
                    MPI_COMM_WORLD, &lcomm, &fileptr, ...);
...
/* Write */
if(sion_ensure_free_space(sid, nbytes)) {
    fwrite(bindata, 1, nbytes, fileptr);
}
...
/* Collective Close */
sion_parclose_mpi(sid);
```

- Writing more data as defined at open call
- SIONlib moves forward to next chunk, if data too large for current block

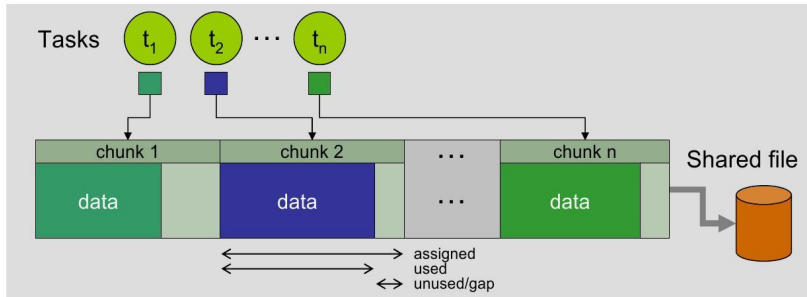
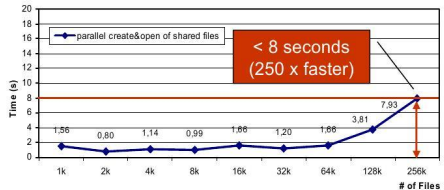
SIONlib in a NutShell: Wrapper function

```
/* Collective Open */
nfiles=1; chunksize=nbytes;
sid=sion_paropen_mpi( filename, "bw", &nfiles, &chunksize,
                    MPI_COMM_WORLD, &lcomm, &fileptr, ...);
...
/* Write */
sion_fwrite(bindata, 1, nbytes, sid);
...
/* Collective Close */
sion_parclose_mpi(sid);
```

- Includes check for space in current chunk
- parameter of fwrite: fileptr → sid

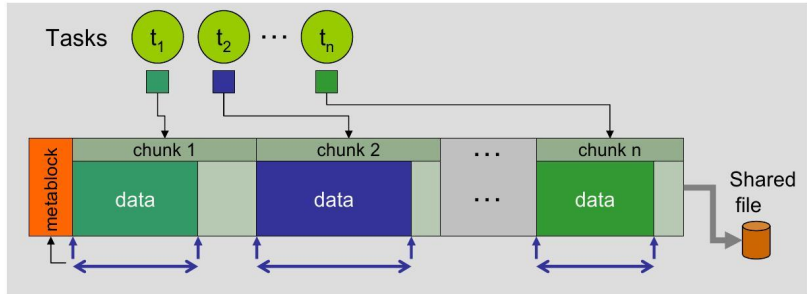
File Format (1): a single shared file

- create and open fast,
- simplified file handling
- logical partitioning required



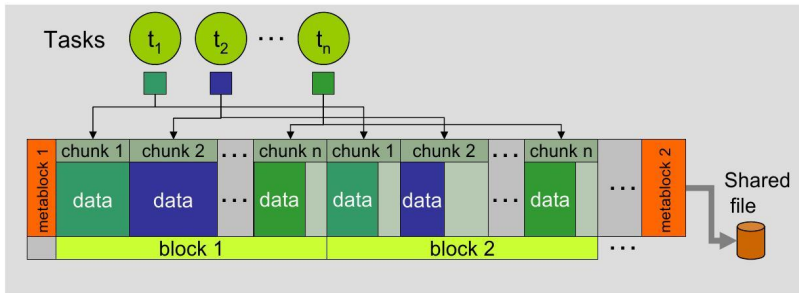
File Format (2): Meta data

- Offset and data size per task
- Tasks have to specify chunk size in advance
- Data must not exceed chunk size



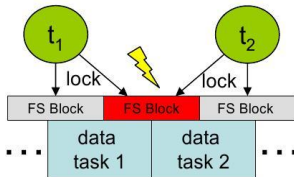
File Format (3): Multiple blocks of chunks

- Enhancement: define blocks of chunks
- Metadata now with variable length ($\#task * \#blocks$)
- Second metadata block at the end
- Data of one block does not exceed chunk size



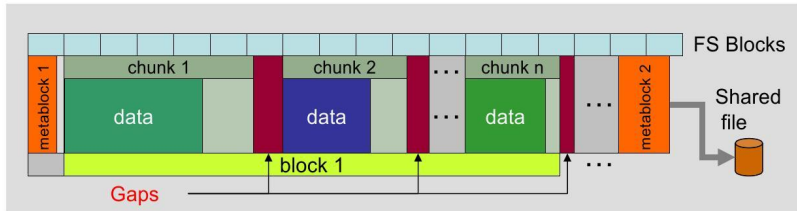
File Format (4): Alignment to block boundaries

- Contention: writing to same file-system block in parallel



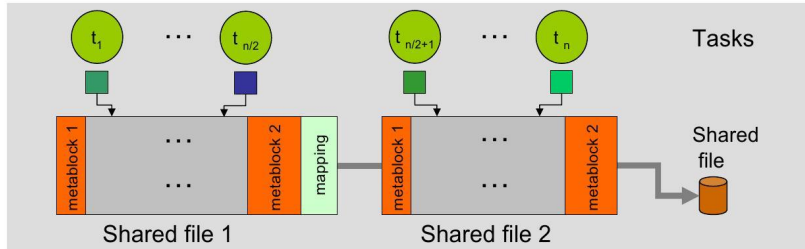
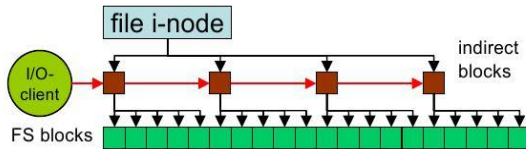
#tasks	data size	blksize	write bandwidth
32768	256 GB	aligned	3650.2 MB/s
32768	256 GB	not aligned	1863.8 MB/s

Jugene (JSC, IBM Blue Gene/P, GPFS, fs:work)



File Format (5): multi-physical files

- Variable number of underlying physical files
- Bandwidth degradation GPFS by using single shared files



Version, Download, Installation

- Version: 1.2.2 (stable), 1.3p2 (new release candidate)
- Version: file format: 4
- Open-Source License, Registration
<http://www.juelich.de/jsc/sionlib>
- Installation: configure; make; make test; make install
- Modules on Jugene:

```
jugene> module avail
----- /usr/local/modulefiles/I0 -----
 hdf5/1.8.4_450(default) sionlib/1.1.9          sionlib/1.3.2
 hdf5/1.8.4_450d       sionlib/1.2.2(default)
 hdf5/1.8.4_ppc        sionlib/1.3.1
```

- Modules on Juropa:

```
juropa> module avail
----- /usr/local/modulefiles/I0 -----
 sionlib/1.2.2(default)
```


Compiling and Linking own Application

- Include file: `#include "sion.h"`
- The installation of sionlib builds (at least) two libraries:
 - `libsionxxx.a`: the parallel libraries currently supporting MPI
 - `libsionserxxx.a`: serial version of sionlib containing all function for the serial API of sionlib
 - `xxx` could be an extensions for precision ('_32', '_64') cross compiling ('fe') or Compiler ('gcc').
- Script: `sionconfig`: prints for each combination of option correct option for compiling (`-cflags`) or linking (`-libs`):

```
usage: sionconfig [--be] [--fe] [--32|--64] [--gcc] [--for]
               [--ser|--mpi] (--cflags|--libs|--path)
```

- Example: (Makefile)

```
LDFLAGS += '../bin/sionconfig --libs --mpi -be'
CFLAGS += '../bin/sionconfig --cflags --mpi -be'
```

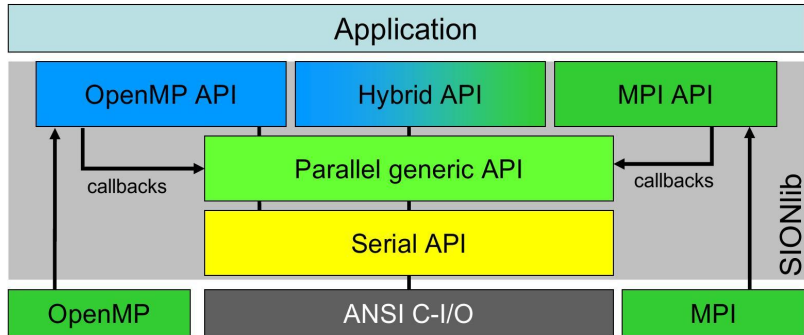
System I/O-Interfaces used by SIONlib

- Under Unix/Linux available: C-Ansi and POSIX
- POSIX Interface
 - `open()`, `write()`, `read()`, `write()`
 - unbuffered, direct access to file
 - File Descriptor: Integer
- ANSI-C
 - `fopen()`, `fwrite()`, `fread()`, `fwrite()`
 - open files and associate a stream with it
 - typically memory buffer of file system block size
 - buffer small consecutive reads and writes
 - File Pointer: `FILE *`
- Fortran Interface: unformatted I/O
 - uses typically internally Posix (or Ansi-C)
 - files opened in C cannot directly accessed from Fortran (mix languages)

SIONlib datatypes

- only used for parameters of SION function calls
- data written to or read from file is a byte stream and need not to be declared by special data types
- `sion_int32`
 - 4-byte signed integer (C)
 - INTEGER*4 (Fortran)
- `sion_int64`
 - 8-byte signed integer (C)
 - INTEGER*8 (Fortran)
 - Typically used for all parameters which could be used to compute file positions

SIONlib: Architecture



Outline

Introduction

Interface

- General Parameters
- Open/Close (Parallel)
- Open/Close (Serial)
- Read/Write
- Get Information
- Seek, Utility Functions

Example

Tools

SIONlib API Overview: Open, Close

- Parallel Interface, using MPI

`sion_paropen_mpi`, `sion_parclose_mpi`

- Parallel Interface, using OpenMP

`sion_paropen_omp`, `sion_parclose_omp`

- Parallel Interface, using MPI+OpenMP

`sion_paropen_omp`, `sion_parclose_omp`

- Serial Interface:

`sion_open`, `sion_open_rank`

`sion_close`

SIONlib API Overview: Read, Write

- Read Data:
 - `sion_fread` (SION, internal check of EOF)
 - `fread()` (Ansi-C)
 - `read()` (Posix)
 - `sion_feof` (Check EOF in chunk)
- Write Data:
 - `sion_fwrite` (SION, internal checks, e.g. chunk size)
 - `fwrite()` (Ansi-C)
 - `write()` (Posix)
 - `sion_flush` (flushes data, updates internal meta data)
 - `sion_ensure_free_space` (Check space in chunk)

SIONlib API Overview: Get Information I

- Get File Pointer for task:
 - `sion_get_fp` (Ansi-C)
 - `sion_get_fd` (Posix)
- Byte order (big(1) or little(0) endian)
 - `sion_get_file_endianness` (Endianness of File)
 - `sion_get_endianness` (Endianness of current system)
- File state
 - `sion_get_bytes_written` (Total number for task written)
 - `sion_get_bytes_read` (Total number for task read)
 - `sion_bytes_avail_in_chunk` (Rest in chunk)
 - `sion_get_position` (Position in file)

SIONlib API Overview: Get Information II

- Multi-physical-file
 - `sion_get_mapping` (Mapping of global task to file and local task, can be used only on task 0 in parallel-mode)
 - `sion_get_number_of_files` (total number of files)
 - `sion_get_filename` (number of file for this task)
- Serial mode: Get information about all tasks
 - `sion_get_locations` (returns pointer to internal chunk description arrays)
 - `sion_is_serial_opened` (indicator for open mode)
- Version
 - `sion_get_version` (returns version of library and fileformat)

SIONlib API Overview: Seek, Utility functions

- Change Position in SION file:
 - `sion_seek` (parallel mode)
 - `sion_seek_fp` (serial mode, change of file pointer possible)
- Utilities:
 - `sion_swap` (change endianness of data in memory)
 - `_sion_file_stat_file` (wrapper for `stat()`, large file support)
- Experimental
 - `sion_coll_fwrite_mpi` (collective write)
 - `sion_coll_fread_mpi` (collective read)

SIONlib parameter of open calls I

- `fname` (file name)
 - character string describing Path and file name
 - will not be extended by SION-specific postfix
 - multiple physical files are generated:
 - first file: `file_name`
 - all other files: `file_name` + “.” + 6-digit-number (000001 ...)
 - all commands and function call uses the base name
- `file_mode`
 - must specify at least one of the following
 - `w,wb,bw` – (write block), Create a new SION file, open for write; overwrite if existing
 - `r,rb,br` – (read block), Open existing SION file for reading
 - `posix` – use internally POSIX interface for file access, otherwise C-ANSI
 - multiple parameter: comma-separated

SIONlib parameter of open calls II

- **sid**
 - `sid = sion_paropen...()` (C)
 - `FSION_PAROPEN_MPI(... , sid)` (Fortran)
 - unique integer value, referring internally to data structure associated to SION file (internal file handle)
 - allows multiple simultaneous opened files
 - C: return code, Fortran: last parameter of open call
 - integer filehandle for Fortran necessary
- **chunksize**
 - Pointer of type `sion_int64*` (C)
 - size in bytes of data written by this tasks
 - could be differ from task to task
 - must be set if open for writing
 - will increased internally to a multiple of the next file system size

SIONlib parameter of open calls III

- `fsblksize`
 - size of file system block in bytes
 - write-mode: will be detected by SIONlib if set to -1
 - read-mode: file system block size at write time
- `newfname`
 - file name of physical file assigned to this task

Collective Open (MPI)

C/C++

```
int sion_paropen_mpi( char *fname, const char *file_mode,
                     int *numFiles,
                     MPI_Comm gComm, MPI_Comm *lComm,
                     sion_int64 *chunksize,
                     sion_int32 *fsblksize,
                     int *globalrank,
                     FILE **fileptr,
                     char **newfname);
```

- Open a SION file in parallel for writing or reading data
- Collective call, call from each task at the same time
- Accesses one or more physical files of a logical SION file
- Parameter are “by reference” to pass back information in write open mode

Collective Open (MPI)

Fortran

```
FSION_PAROPEN_MPI ( FNAME, FILE_MODE, NUMFILES,  
                   GCOMM, LCOMM, CHUNKSIZE, FSBLKSIZE,  
                   GLOBALRANK, NEWFNAME, SID)  
CHARACTER*(*) FNAME, FILE_MODE, NEWFNAME  
INTEGER      NUMFILES, FSBLKSIZE, GLOBALRANK, SID, GCOMM, LCOMM  
INTEGER*8    CHUNKSIZE
```

- Open a SION file in parallel for writing or reading data
- Collective call, call from each task at the same time
- Accesses one or more physical files of a logical SION file
- Parameter are “by reference” to pass back information in write open mode

special parameter of *sion_paropen_mpi* I

- Communicator `gComm`
 - Call is collective over all tasks of this communicator
 - Each task get assigned one chunk of SION file
 - Read: Number of tasks must be equivalent to number tasks written to SION file
- Number of physical files
 - `numfiles`, or -1 if specified by communicator
 - `lComm` or `MPI_Comm_null`
 - Read-mode: parameters will be set by open call
- `globalrank`
 - rank of task in global communicator `gComm`

Collective Close (MPI)

C/C++

```
int sion_parclose_mpi ( int sid );
```

- Closes a SION file in parallel on all tasks
- Collective call, call from each task of (gComm) at the same time
- Meta data will be collected from each tasks
- Meta data blocks of SION file will be written in this call
- Currently no fault tolerant handling of the meta-data, call has to executed

Collective Close (MPI)

Fortran

```
FSION_PARCLOSE_MPI (SID, IERR)  
INTEGER            SID, IERR
```

- Closes a SION file in parallel on all tasks
- Collective call, call from each task of (gComm) at the same time
- Meta data will be collected from each tasks
- Meta data blocks of SION file will be written in this call
- Currently no fault tolerant handling of the meta-data, call has to executed

Collective Open (OpenMP)

C/C++

```
int sion_paropen_omp (    char          *fname,  
                        const char      *file_mode,  
                        sion_int64      *chunksize,  
                        sion_int32      *fsblksize,  
                        int             *globalrank,  
                        FILE            **fileptr,  
                        char            **newfname);
```

- Open a SION file in parallel for writing or reading data
- Collective call, call has to be called inside a parallel region
- SION file consists of only one physical file
- Parameter are “by reference” to pass back information in write open mode
- Thread-number: `globalrank`

Collective Open (OpenMP)

Fortran

```
FSION_PAROPEN_OMP ( FNAME, FILE_MODE,  
                   CHUNKSIZE, FSBLKSIZE,  
                   GLOBALRANK, NEWFNAME, SID)  
CHARACTER*(*) FNAME, FILE_MODE, NEWFNAME  
INTEGER      FSBLKSIZE, GLOBALRANK, SID  
INTEGER*8    CHUNKSIZE
```

- Open a SION file in parallel for writing or reading data
- Collective call, call has to be called inside a parallel region
- SION file consists of only one physical file
- Parameter are “by reference” to pass back information in write open mode
- Thread-number: `globalrank`

Collective Close (OpenMP)

C/C++

```
int sion_parclose_omp ( int sid );
```

- Closes a SION file in parallel on all threads
- Collective call, call has to be called inside a parallel region
- Meta data will be collected from each thread
- Meta data blocks of SION file will be written in this call
- Currently no fault tolerant handling of the meta-data, call has to be executed

Collective Close (OpenMP)

Fortran

```
FSION_PARCLOSE_OMP (SID, IERR)  
INTEGER            SID, IERR
```

- Closes a SION file in parallel on all threads
- Collective call, call has to be called inside a parallel region
- Meta data will be collected from each thread
- Meta data blocks of SION file will be written in this call
- Currently no fault tolerant handling of the meta-data, call has to be executed

Collective Open (MPI+OpenMP)

C/C++

```
int sion_paropen_ompi( char *fname, const char *file_mode,  
                      int *numFiles,  
                      MPI_Comm gComm, MPI_Comm *lComm,  
                      sion_int64 *chunksize,  
                      sion_int32 *fsblksize,  
                      int *globalrank,  
                      FILE **fileptr,  
                      char **newfname);
```

- Open a SION file in parallel for writing or reading data
- Collective call, call from each task/thread at the same time, call has to be called inside a parallel region
- Parameter and further description see MPI and OpenMP functions

Collective Open (MPI+OpenMP)

Fortran

```
FSION_PAROPEN_OMPI ( FNAME, FILE_MODE, NUMFILES,  
                     GCOMM, LCOMM, CHUNKSIZE, FSBLKSIZE,  
                     GLOBALRANK, NEWFNAME, SID)  
CHARACTER*(*) FNAME, FILE_MODE, NEWFNAME  
INTEGER      NUMFILES, FSBLKSIZE, GLOBALRANK, SID, GCOMM, LCOMM  
INTEGER*8    CHUNKSIZE
```

- Open a SION file in parallel for writing or reading data
- Collective call, call from each task/thread at the same time, call has to be called inside a parallel region
- Parameter and further description see MPI and OpenMP functions

Collective Close (MPI+OpenMP)

C/C++

```
int sion_parclose_ompi ( int sid );
```

- Closes a SION file in parallel on all tasks/threads
- Collective call, call from each task of (gComm) at the same time, call has to be called inside a parallel region
- Meta data will be collected from each tasks
- Meta data blocks of SION file will be written in this call
- Currently no fault tolerant handling of the meta-data, call has to executed

Collective Close (MPI+OpenMP)

Fortran

```
FSION_PARCLOSE_OMPI (SID, IERR)  
INTEGER              SID, IERR
```

- Closes a SION file in parallel on all tasks/threads
- Collective call, call from each task of (gComm) at the same time, call has to be called inside a parallel region
- Meta data will be collected from each tasks
- Meta data blocks of SION file will be written in this call
- Currently no fault tolerant handling of the meta-data, call has to executed

Serial Open

C/C++

```
int sion_open (    char *fname,  
                  const char* file_mode,  
                  int *ntasks, int *nfiles,  
                  sion_int64 **chunksizes,  
                  sion_int32 *fsblksize,  
                  int **globalranks,  
                  FILE **fileptr);
```

- Open a SION file in serial mode
- all chunks of all tasks could be selected, via `sion_seek_fp`
- multi-physical-file could be handled
- designed to use in serial pre- and post-processing tools
- reads all meta-data of all tasks into memory

Serial Open

Fortran

```
FSION_OPEN ( FNAME, FILE_MODE, NTASKS, NUMFILES,  
             CHUNKSIZES, FSBLKSIZE,  
             GLOBALRANKS, SID)  
CHARACTER*(*) FNAME, FILE_MODE  
INTEGER      NUMFILES, NTASKS, FSBLKSIZE, SID  
INTEGER      GLOBALRANKS(ntasks)  
INTEGER*8    CHUNKSIZES(ntasks)
```

- Open a SION file in serial mode
- all chunks of all tasks could be selected, via `sion_seek_fp`
- multi-physical-file could be handled
- designed to use in serial pre- and post-processing tools
- reads all meta-data of all tasks into memory

Serial Open for one Rank

C/C++

```
int sion_open_rank(    char *fname,  
                      const char *file_mode,  
                      sion_int64 *chunksize,  
                      sion_int32 *fsblksize,  
                      int *rank,  
                      FILE **fileptr);
```

- Open SION file for one rank in serial mode
- multi-physical-file could be handled
- designed to use in parallel program if collective open/close is not possible
- reads only meta-data of this task into memory

Serial Open for one Rank

Fortran

```
FSION_OPEN_RANK ( FNAME, FILE_MODE,  
                  CHUNKSIZE, FSBLKSIZE,  
                  RANK, SID)  
CHARACTER*(*) FNAME, FILE_MODE  
INTEGER      FSBLKSIZE, SID, RANK  
INTEGER*8    CHUNKSIZE
```

- Open SION file for one rank in serial mode
- multi-physical-file could be handled
- designed to use in parallel program if collective open/close is not possible
- reads only meta-data of this task into memory

Serial Close

C/C++

```
int sion_close ( int sid );
```

- Closes a SION file in serial mode
- Meta data blocks of SION file will be written in this call
- Currently no fault tolerant handling of the meta-data, call has to be executed

Serial Close

Fortran

```
FSION_CLOSE (SID, IERR)  
INTEGER      SID, IERR
```

- Closes a SION file in serial mode
- Meta data blocks of SION file will be written in this call
- Currently no fault tolerant handling of the meta-data, call has to be executed

Read Data

C/C++

```
size_t sion_fread(void *ptr,  
                 size_t size, size_t nmemb,  
                 int sid);
```

- Read $\text{size} \times \text{nmemb}$ bytes from current position in chunk
- Internally this function reads in a while loop until all data is read from file. Reading more data than stored in one chunk is with this wrapper possible.
- Returns number of bytes read
- Wrapper for `sion_read`, `fsion_fread` will be implemented

Read Data

Fortran

```
FSION_READ (DATA, SIZE, NMEMB, SID, IERR)  
INTEGER    SIZE, NMEMB, SID, IERR
```

- Read $\text{size} * \text{nmemb}$ bytes from current position in chunk
- Internally this function reads in a while loop until all data is read from file. Reading more data than stored in one chunk is with this wrapper possible.
- Returns number of bytes read
- Wrapper for `sion_read`, `fsion_fread` will be implemented

End of File

C/C++

```
int sion_feof(int sid);
```

- Equivalent to POSIX feof which cannot be used for share SION files
- Internally this function flushes all buffer and checks current positions against chunk boundaries
- Moves file pointer to next chunk if end of current chunk is reached
- The function is a task local function, which can be called independently from other MPI tasks.
- Returns 1 if pointer is behind last byte of data for this task

End of File

Fortran

```
FSION_FEOF (SID, EOF)  
INTEGER    SID, IERR
```

- Equivalent to POSIX feof which cannot be used for share SION files
- Internally this function flushes all buffer and checks current positions against chunk boundaries
- Moves file pointer to next chunk if end of current chunk is reached
- The function is a task local function, which can be called independently from other MPI tasks.
- Returns 1 if pointer is behind last byte of data for this task

Write Data

C/C++

```
size_t sion_fwrite(const void *data,  
                  size_t size, size_t nitems,  
                  int sid);
```

- Write $\text{size} \times \text{nitems}$ bytes to chunk, beginning from current position
- Internally this function checks with `sion_ensure_free_space` if enough space is available.
- returns number of bytes written
- wrapper for `sion_write`, `fsion_fwrite` will be implemented

Write Data

Fortran

```
FSION_WRITE (DATA, SIZE, NMEMB, SID, IERR)  
INTEGER      SIZE, NMEMB, SID, IERR
```

- Write $\text{size} \times \text{nmemb}$ bytes to chunk, beginning from current position
- Internally this function checks with `sion_ensure_free_space` if enough space is available.
- returns number of bytes written
- wrapper for `sion_write`, `fsion_fwrite` will be implemented

Flush Data

C/C++

```
int sion_flush(int sid);
```

- After writing of data this function updates internal data structures to new file position
- To obtain new file position a POSIX flush will be used which could be time consuming

Flush Data

Fortran

```
FSION_FLUSH (SID, IERR)  
INTEGER      SID, IERR
```

- After writing of data this function updates internal data structures to new file position
- To obtain new file position a POSIX flush will be used which could be time consuming

Ensure Free Space in Chunk

C/C++

```
int sion_ensure_free_space(int sid, sion_int64 bytes);
```

- Ensures that there is enough space available for writing
- A new chunk will be allocated if bytes could not be written in the current chunk
- The function is a task local function, which can be called independently from other MPI tasks
- The function moves in some cases the filepointer to a new position and flushes also the local filepointer
- returns 1 if space could ensured, there is currently no indicator if a new chunk was allocated

Ensure Free Space in Chunk

Fortran

```
FSION_ENSURE_FREE_SPACE (SID, BYTES, IERR)  
INTEGER      SID, IERR  
INTEGER*8    BYTES
```

- Ensures that there is enough space available for writing
- A new chunk will be allocated if bytes could not be written in the current chunk
- The function is a task local function, which can be called independently from other MPI tasks
- The function moves in some cases the filepointer to a new position and flushes also the local filepointer
- returns 1 if space could ensured, there is currently no indicator if a new chunk was allocated

Get File Pointer

```
FILE * sion_get_fp(int sid); /* Ansi-C */  
int    sion_get_fd(int sid); /* POSIX */
```

- Returns Ansi-C file pointer (`_fp`) or
- Returns POSIX File descriptor
- File pointer/descriptor corresponds to physical file containing data of current task
- SION file must be opened with corresponding option
- the POSIX file descriptor can be obtained from a Ansi-C file pointer: `fd = fileno(fileptr)`
- Ansi-C file pointer can be obtained from a POSIX file descriptor: `fileptr = fdopen(fd, "r")`

Get Byte Ordering (Endianness)

```
int sion_get_file_endianness(int sid);  
int sion_get_endianness();
```

- return endianness (1-¿ big endian, 0 -¿ little endian)
- for current file (sid), or
- for current runtime environment
- bytes has to be reordered if:
`sion_get_file_endianness() != sion_get_endianness()`
- Utility for reordering: see `sion_swap`
- Currently no Fortran API ! TBD: implementation

Seek: Change File Position

C/C++

```
int sion_seek( int sid,
               int rank,
               int chunknr,
               sion_int64 posinchunk );
```

- Sets the file pointer to a new position
- Seek parameters:
 - `rank`: rank number (0,...), or SION_CURRENT_RANK
 - `chunknum`: chunk number (0,...), or SION_CURRENT_BLK
 - `posinchunk`: position (0,...), or SION_CURRENT_POS
- In parallel write mode is seeking currently not supported
- For serial opened file please use `sion_seek_fp`, because physical file pointer could change.

Seek: Change File Position

Fortran

```
FSION_SEEK (SID, RANK, CHUNKNUM, POSINCHUNK, IERR)  
INTEGER    SID, RANK, CHUNKNUM, IERR  
INTEGER*8  POSINCHUNK
```

- Sets the file pointer to a new position
- Seek parameters:
 - `rank`: rank number (0,...), or `SION_CURRENT_RANK`
 - `chunknum`: chunk number (0,...), or `SION_CURRENT_BLK`
 - `posinchunk`: position (0,...), or `SION_CURRENT_POS`
- In parallel write mode is seeking currently not supported
- For serial opened file please use `sion_seek_fp`, because physical file pointer could change.

Seek: Change File Position + FilePtr

C/C++

```
int sion_seek_fp( int sid,  
                  int rank,  
                  int chunknr,  
                  sion_int64 posinchunk,  
                  FILE **fileptr );
```

- Sets the file pointer to a new position
- Seek parameters → see sion_seek, in addition:
 - `fileptr`: Ansi-C pointer to file, should be used after seeking instead of `fileptr` of open call
- No Fortran wrapper, `fileptr` unknown in Fortran

Seek: Change File Position + FilePtr

Fortran

```
FSION_SEEK (SID, RANK, CHUNKNUM, POSINCHUNK, IERR)  
INTEGER    SID, RANK, CHUNKNUM, IERR  
INTEGER*8  POSINCHUNK
```

- Sets the file pointer to a new position
- Seek parameters → see `sion_seek`, in addition:
 - `fileptr`: Ansi-C pointer to file, should be used after seeking instead of `fileptr` of `open` call
- No Fortran wrapper, `fileptr` unknown in Fortran

Utility: Swap bytes

C/C++

```
void sion_swap(void *target, void *source,  
               int size, int n, int aflag);
```

- perform byte-order swapping for arrays of `n` units of `size` byte
- bytes are swapped if and only if `aflag == 0`
- data will copied from `source` to `target`
- in-place swapping (`target == source`) is allowed if `target != source`, the buffers must not overlap
- `aflag` could be initialized as follow:
`sion_get_file_endianness() == sion_get_endianness()`

Utility: Swap bytes

Fortran

```
FSION_SWAP (TARGET, SOURCE, SIZE, N, AFLAG, IERR)  
INTEGER      SIZE, N, AFLAG, IERR    ! TBD: implementation
```

- perform byte-order swapping for arrays of `n` units of `size` byte
- bytes are swapped if and only if `aflag` == 0
- data will copied from `source` to `target`
- in-place swapping (`target` == `source`) is allowed if `target` != `source`, the buffers must not overlap
- `aflag` could be initialized as follow:
`sion_get_file_endianness() == sion_get_endianness()`

Outline

Introduction

Interface

Example

Tools

Example Code: sion_par_write (Part 1)

C/C++

```
#include <sion.h>

/* SION parameters */
int      sid, numFiles, globalrank;
MPI_Comm lComm;
sion_int64 chunksize, left, bwrote;
sion_int32 fsblksize;
char      fname[256], *newfname=NULL;
FILE      *fileptr;
/* initialize MPI */
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
/* open parameters */
chunksize = 10;  globalrank = my_rank;
strcpy(fname, "parfile.sion");
numFiles   = 1;  fsblksize   = -1;
```

Example Code: sion_par_write (Part 1)

Fortran

```
! SION parameters
integer*8          :: chunksize
integer            :: gComm,lComm,sid,globalrank,ierr
integer            :: fsblksize, nfiles
character(len=255) :: filename = 'test_sionfile.dat'
character(len=255) :: newfname
integer*4,dimension(:),allocatable :: buffer

! MPI initialization
call MPI_Init(ierr)
call MPI_Comm_size(MPI_COMM_WORLD,nranks,ierr)
call MPI_Comm_rank(MPI_COMM_WORLD,my_rank,ierr)

! create a new file
gcomm=MPI_COMM_WORLD
globalrank=my_rank
fsblksize=-1
chunksize=10
nfiles=1
```

Example Code sion_par_write (Part 2)

C/C++

```
/* create a new file */
sid = sion_paropen_mpi(fname, "bw", &numFiles,
                      MPI_COMM_WORLD, &lComm,
                      &chunksize, &fsblksize,
                      &globalrank,
                      &fileptr, &newfname);

/* write buffer to file */
left=chunksize; p = (char *) fname;
while (left > 0) {
    sion_ensure_free_space(sid, left);
    bwrote = fwrite(p, 1, left, fileptr);
    left -= bwrote; p += bwrote;
}
/* close file */
sion_parclose_mpi(sid);
/* finalize MPI */
MPI_Finalize();
```

Example Code sion_par_write (Part 2)

Fortran

```
call fsion_paropen_mpi(trim(filename),'bw', &
                        nfiles, gComm, &
                        lComm, chunksize, &
                        fsblksize, globalrank, &
                        newfname, sid)

! write buffer to file
call fsion_write(buffer, 4, veclen, sid, ierr)

! close file
call fsion_parclose_mpi(sid, ierr)

! MPI finalization
call MPI_Finalize(ierr)
```

Example: Blue Gene/P I/O-node Task Mapping

- Blue Gene/P CPU-nodes are connected to I/O-nodes (Jugene: 128 CPU-nodes : 1 I/O-Node)
- Good performance: one physical file per I/O-node
- Special MPI Communicator containing all tasks connected to the same I/O-Node (Pset)

```
/* communicator consists of all task  
   working with the same I/O-node */  
MPI_Comm commSame;  
MPIX_Pset_same_comm_create(&commSame);  
MPI_Comm_size(commSame, &sizeSame);  
MPI_Comm_rank(commSame, &rankSame);  
  
/* create a new file */  
sid = sion_paropen_mpi(fname, "bw", &numFiles,  
                       MPI_COMM_WORLD, &commSame,  
                       &chunksize, &fsblksize,  
                       &globalrank, &fileptr, &newfname);
```


Outline

Introduction

Interface

Example

Tools

Managing SION files
Parallel Benchmark

Tools: sionsplit

- Generates task-local files from SION-file
- Usage: `sionsplit [options] sionfile prefix`
- Options:

<code>prefix</code>	directory and/or filename-prefix for task-local files
<code>[-v]</code>	verbose mode
<code>[-g]</code>	use global rank for numbering files
<code>[-d <num>]</code>	number of digits for filename generation (default 5)

Tools: sioncat

- Extract all or selected data from a SION file
- Usage: `sioncat [options] sionfile`
- Options:

<code>[-v]</code>	verbose mode
<code>[-t <tasknum>]</code>	write only data of task <tasknum>
<code>[-b <blknum>]</code>	write only data of block <blknum>
<code>[-o <outfile>]</code>	file data will be written to, if not specified stdout will be used

Tools: siondump

- Print meta data information of a SION file
- Usage: `siondump [options] sionfile`
- Options:

<code>[-a]</code>	print all information about all blocks
<code>[-m]</code>	print all mapping information
<code>[-l]</code>	print all sizes in number of bytes
<code>[-v]</code>	verbose mode

Tools: siondefrag

- Generates new SION file from existing one,
- changing the underlying file system block size
- can be used to remove empty gaps, caused by
 - not completely filled chunks
 - alignment to file system blocks
- Usage: `siondefrag [options] sionfile new_sionfile`
- Options:

```
[-Q <fsblksize>]  filessystem blocksize for new sion file  
                   in MB          (default from input file)  
[-q <fsblksize>]  filessystem blocksize for new sion file  
                   in bytes
```

Tools: partest, Parallel I/O benchmark I

- Usage: `partest [options]`
- Options (file settings):

```
[-f filename]      (--filename[=])  filename of direct access file
[-n <numfiles>]    (--numfiles[=])  number of files file
[-r <chunksize>]   (--chunksize[=]) sion chunk size (*)
[-q <fsblksize>]   (--fsblksize[=]) size of filesystem blocks (*)
```

- Options (test configuration):

```
[-T <type>]        (--testtype[=])  testtype (0:SION, collective)
                                   (1:SION, independent read)
                                   (2:MPI IO) (3:Task-Local)
[-b <bufsize>]      (--bufsize[=])   blocksize written by ONE fwrite (*)
[-g <totalsize>]     (--totalsize[=])  global total size of data written(*)
[-s <localsize>]     (--localsize[=])  size of local data for each task(*)
[-F <factor>]        (--factor[=])    random factor (0.0 to 1.0, def: 0.0)
[-R (0|1)]           (--read[=])      switch read  off/on
[-W (0|1|2)]         (--write[=])     switch write off, on, or 2x write

(*) Size Formats: <d>[g,G,Gb,GB, m,M,Mb,MB, k,K,Kb,KB, GiB, MiB, KiB ]
```

Tools: partest, Parallel I/O benchmark II

- Options (test specific configuration):

<code>[-v]</code>	<code>(--verbose[=] (0 1))</code>	verbose print info for each task
<code>[-C]</code>	<code>(--nochecksum[=] (0 1))</code>	suppress checksum
<code>[-d]</code>	<code>(--debugtask[=] (0 1))</code>	debug task 0
<code>[-D]</code>	<code>(--Debugtask[=] (0 1))</code>	debug task n
<code>[-L]</code>	<code>(--posix[=] (0 1))</code>	use POSIX calls instead of ANSI calls
<code>[-M]</code>	<code>(--collwrite[=] (0 1))</code>	use collective write if possible
<code>[-m]</code>	<code>(--collread[=] (0 1))</code>	use collective read if possible
<code>[-Z <offset>]</code>	<code>(--taskoffset[=])</code>	shift tasks numbering for reading by offset to omit data caching of file-system (0)
<code>[-O <bytes>]</code>	<code>(--byteoffset[=])</code>	start offset, write <bytes> first before using blksize (0)
<code>[-j <#tasks>]</code>	<code>(--serialized[=])</code>	serialize I/O, only I/O of #tasks are running in parallel -1 -> all tasks in parallel, -2 -> use transactions, def: -1
<code>[-X]</code>	<code>(--unlinkfiles[=] (0 1))</code>	remove files after test

Tools: partest, Parallel I/O benchmark III

- Options (Blue Gene/L, Blue Gene/P):

```
[ -P ]                ( --bgionode [=] ( 0 | 1 ) )    order tasks by BG I/O-node  
[ -p <numtasks> ]    ( --bgtaskperionode [=] )        number of tasks  
                                                            per BG I/O-node (default: all)
```

- Options (MPI-I/O Hints):

```
[ -w ]                ( --hintlargeblock [=] ( 0 | 1 ) )    IBM, Large Block IO  
[ -Q <size> ]        ( --hintiobufsize [=] )                IBM, IO bufsize in KB  
[ -x ]                ( --hintsparseaccess [=] ( 0 | 1 ) )  IBM, sparse access
```


Outline

Introduction

Interface

Example

Tools

Exercise: Parallel Write/Read

- Create two parallel application (C, Fortran):
 - 1** Write:
 - Creating a SION data set, where each task writes local data to a the corresponding chunk of the SION-file
 - A local vector of 10000 integers should be allocated and initialized with the task number
 - Each task should write the vector to the SION data set
 - 2** Read
 - Read the data of the corresponding chunk into memory
 - and check if the data is consistent (task number)
- Run `siondump` on the SION-file to check the meta-data
- Create with `siondefrag` a dense version of the SION-file, and check again the meta-data of the new file
- Extract the chunks of the SION file into task-local files
- check if data is written with same endianness, swap if necessary