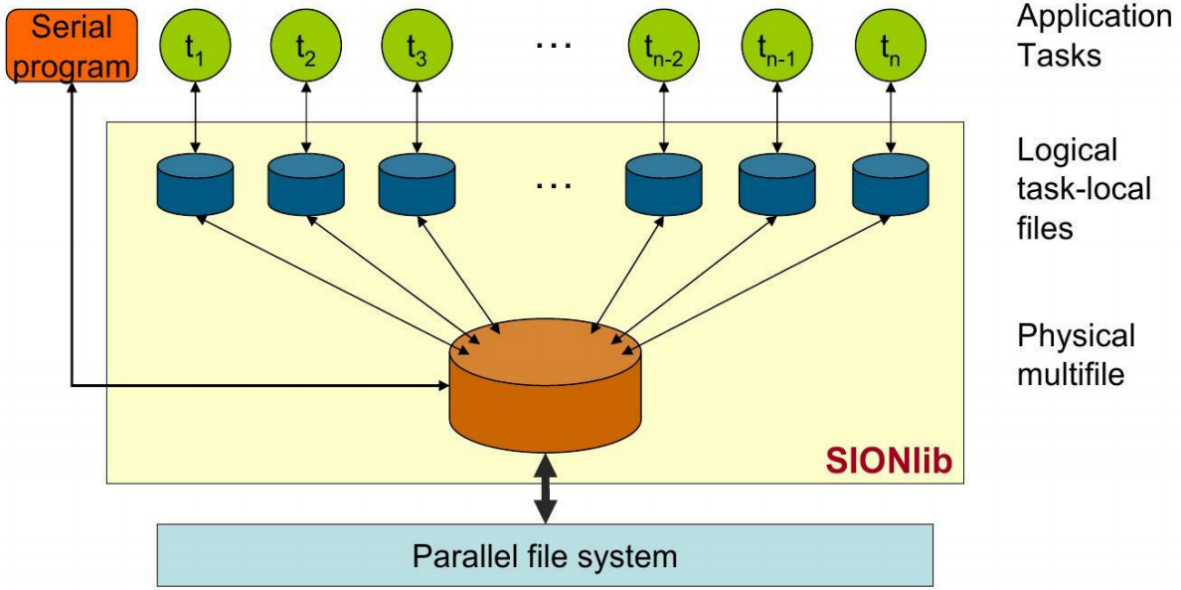
PRACE PATC Course: Advanced Topics in HPC
Topic: SIONLib Library

Sandra Mendez, PhD - HPC Group, LRZ

sandra.mendez@lrz.de

Outline

Motivation



Limitations of Task-Local I/O

- contention at the meta data server
- May degrade the I/O Performance
- complicated file handling (e.g. archive)

Using Shared Files

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

SIONlib: Introduction

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

SIONlib Tutorial:

- https://apps.fz-juelich.de/jsc/sionlib/html/sionlib_tutorial_2013.pdf

SIONlib Download, Documentation, and Information:

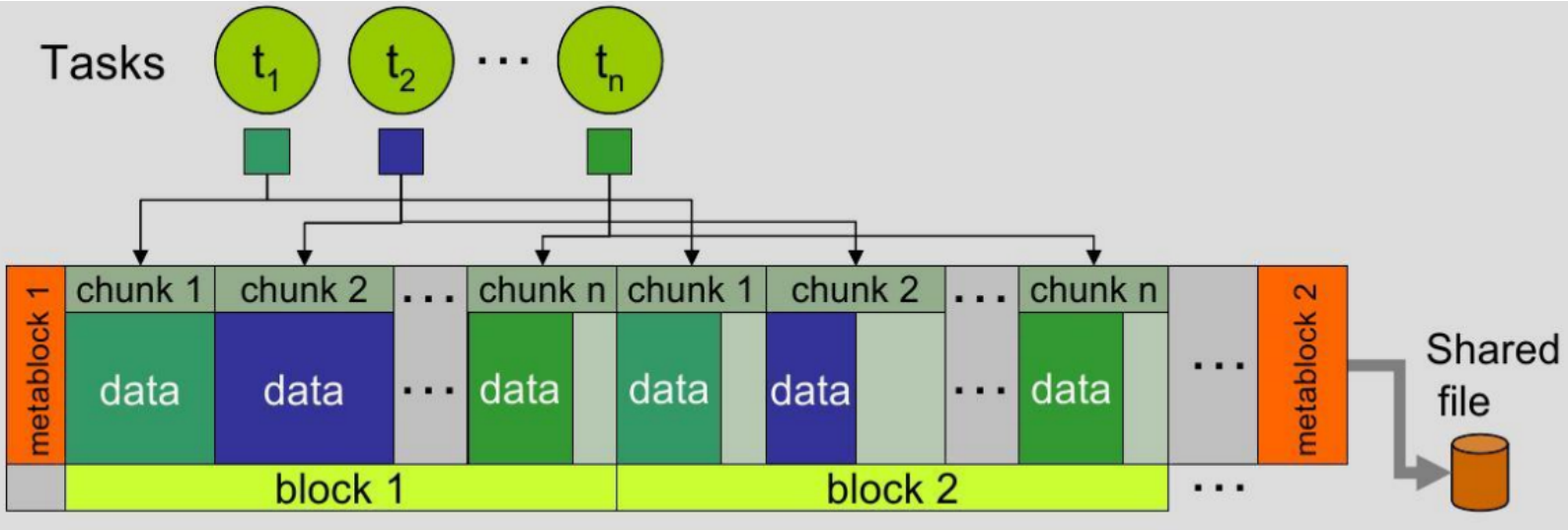
- http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/SIONlib/_node.html

Parallel I/O for Large Scale

- 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 file format (1)



A single shared file: • create and open fast • simplified file handling • logical partitioning required.

Meta data:

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

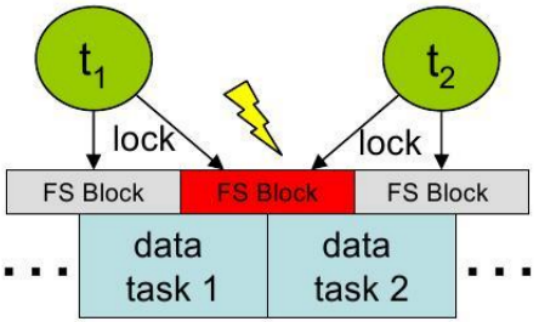
Multiple blocks of chunks:

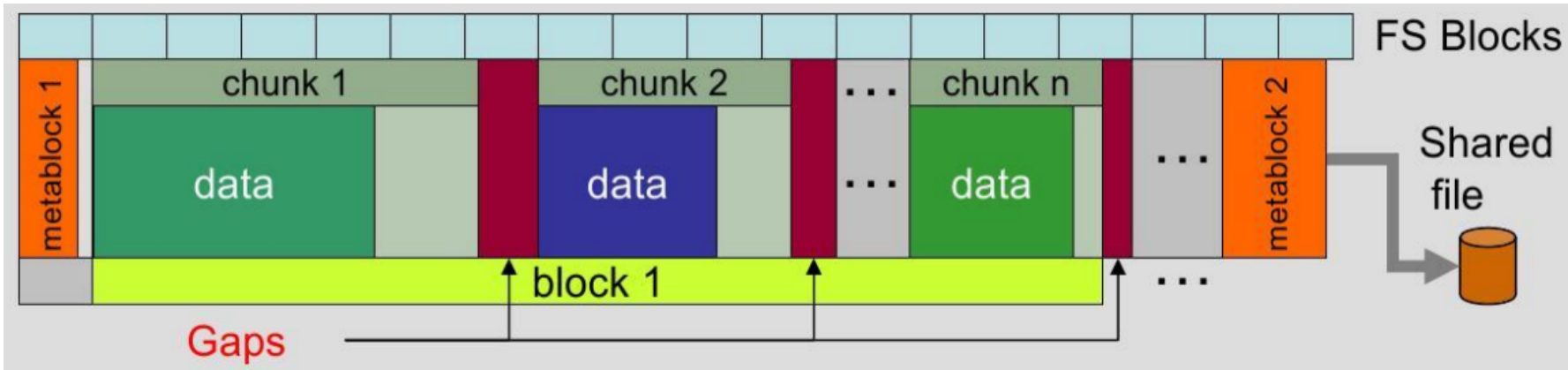
- Enhancement: define blocks of chunks
- Metadata now with variable length (#task * #blocks)
- Second metadata block at the end
- Data of one block does not exceed chunk size

SIONlib file format (2)

Alignment to block boundaries

Contention: writing to same file-system block in parallel



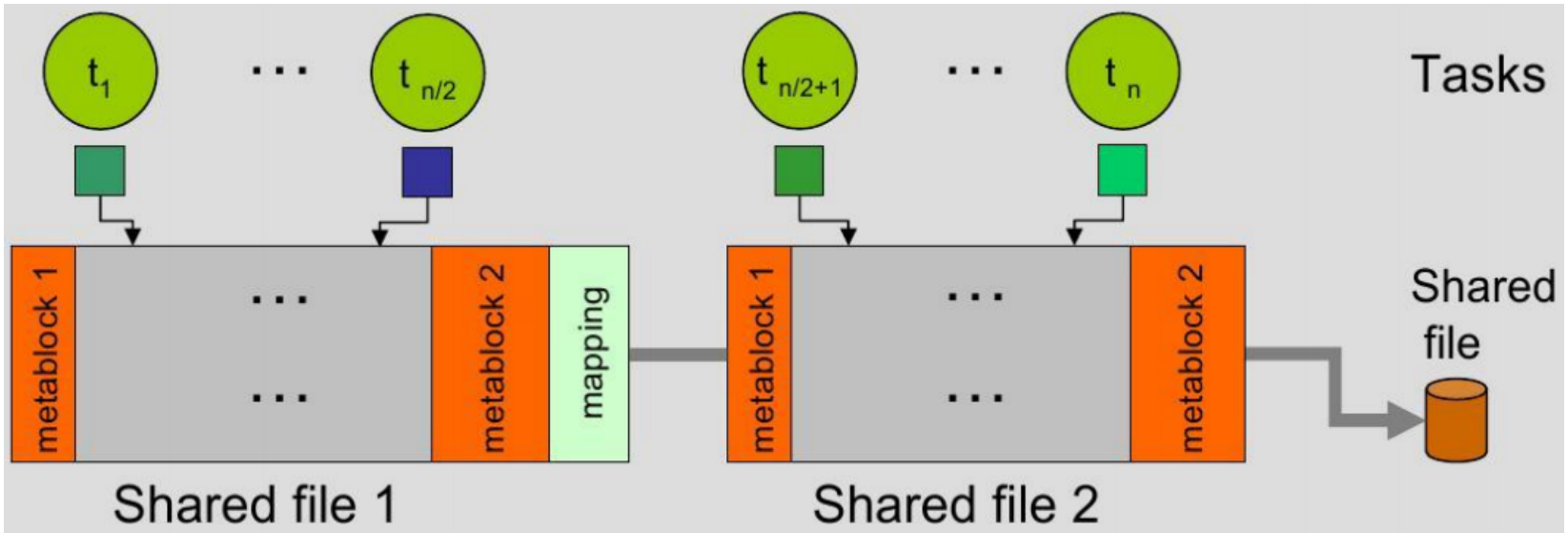


All starting positions of the blocks are aligned to the file system blocksize

SIONlib file format (3)

Multi-physical files

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



SIONlib I/O Interfaces

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

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`

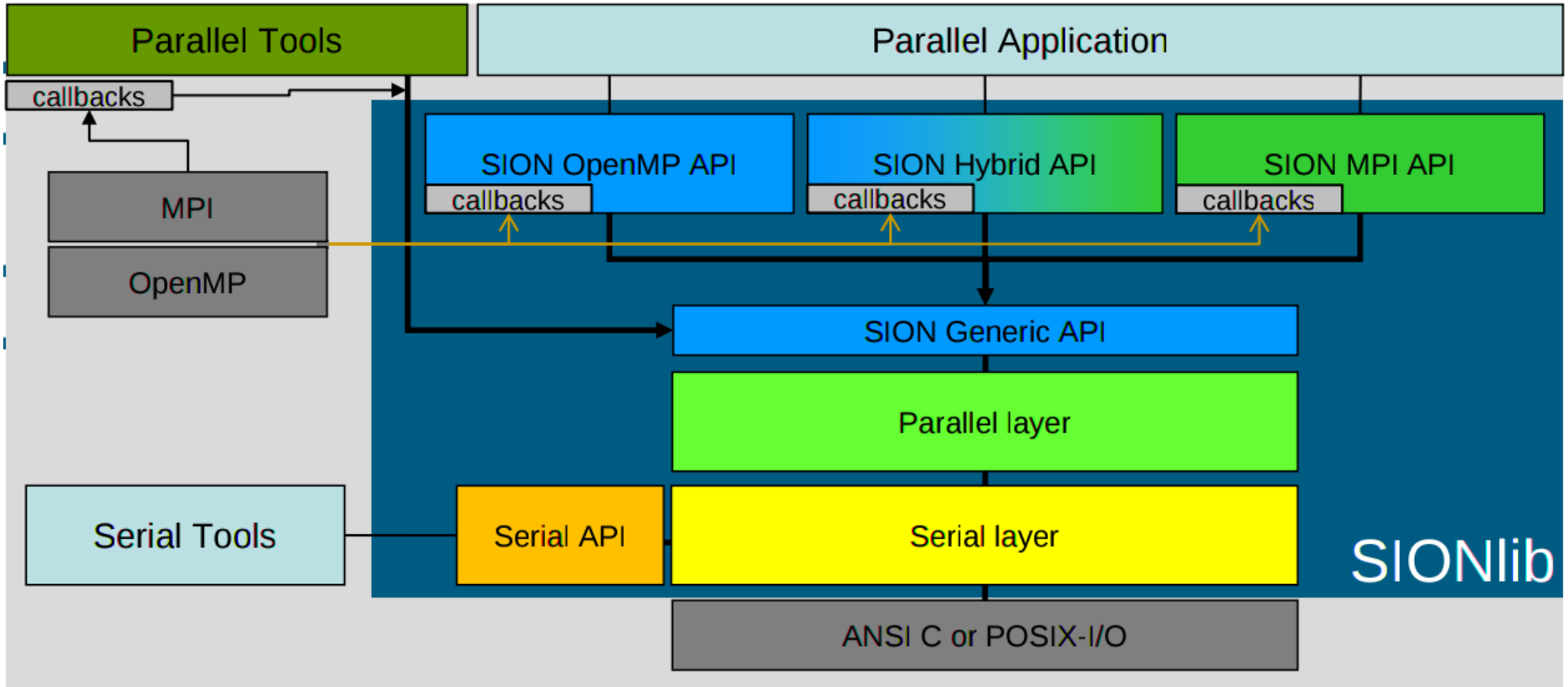
  - 8-byte signed integer (C)
  - INTEGER*8 (Fortran)
  - Typically used for all parameters which could be used to compute file positions

SIONlib: API & Utilities

Parallel Interface	Serial Interface	Common Interface
sion_paropen_mpi()	sion_open()	sion_ensure_free_space()
sion_parclose_mpi()	sion_open_rank()	sion_feof()
	sion_close()	sion_bytes_avail_in_block()
	sion_get_locations()	sion_seek()
		sion_seek_fp()
		sion_fwrite()
		sion_fread()

siondump	Dumping the meta data of a sion file
sionsplit	Splitting one sion file in separate files
siondefrag	De-fragmenting a sion file
sioncat	Extracts all data or data of one tasks
partest	Parallel test of sionlib

SIONlib Layer



SIONlib: Basic example (1)

```
/* 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 bytess

SIONlib: Basic example (2)

```
/* 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: sion_paropen_mpi

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

- **file_mode** like the type parameter of fopen (file mode description)
- **numFiles** number of multi files to use (-1 for automatic choosing from local communicator)
- **gComm** global MPI communicator (typically MPI_COMM_WORLD)
- **lComm** local MPI communicator (= gComm if no adaption to I/O nodes is needed)
- **chunksize** maximum size to be written with single write call
- **fsblksize** file system block size. Must be equal on all processes (-1 for automatic)
- **globalrank** global rank of process any globally unique id for current task.

The file_mode argument

file_mode is a string that contains a comma-separated list of key value pairs

- "w", "bw" or "wb": open file in write mode
- "r", "br" or "rb": open file in read mode
- "ansi" or "posix": use ANSI C or POSIX interface for the file access. ANSI C offers internal buffering and is the default.
- "collective", : use collective operations
- "collectivemerge" or "cmerge": use collective operations with collective merge mode
- "keyval[=MODE]": open file in key value mode. The default is default.
- "endianness=MODE": force the file to be opened with endianness MODE. Possible values are big and little. The current system's endianness is used by default.

Example: 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 to large for current block

Example: 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 $\rightarrow$ sid

SIONlib Summary

- Optimises binary one-file-per-processes approach by usage of a shared container file
- Uses own file format
- More convenient for internal format (Portability is not the main priority)
- Provide a transparent mechanism to avoid file system block contention.
- Part of Tools/Projects:
  - Scalasca: Performance Analysis
  - Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes
  - DEEP-ER: Adaption to new platform and parallelization paradigm

Summary

- I/O Performance results heavily depend on the soft- and hardware architecture of the underlying I/O system and application's I/O Pattern.
- Specialized I/O libraries:
  - may provide more portable way of writing data
  - may reduce metadata load when properly used
- For parallel programs
  - output to separate files for each process: highest throughput, but usually needs post-processing
  - investigate whether MPI IO is more suitable (changing number of tasks between production runs)
- May need to use library/compiler support for conversion
  - if binary files are transferred between different architectures (little vs. big-endian byte order)
  - limitations may apply on file sizes and data types.

Thank you for your attention!