gbpSID Documentation

Release 0.0.1

Gregory B. Poole

Mar 18, 2018

Contents

CHAPTER 1

Installation

To aquire and build this library, you will need to have `git` and `cmake` installed. You may also want one-or-more of the following optional 3rd-party libraries:

- an MPI distribution (eg. OpenMPI)
- CUDA (if you have an NVidia GPGPU installed).

With these installed, you will need to:

1. acquire the code from *GitHub*;
2. configure it with `cmake`; and
3. build it with `make`

Alternatively (an prehaps more-often-than-not), you may want to add *gbpSID* as a submodule to another project. In either case, each of these steps is described in more detail below.

1.1 Acquireing the code

gbpSID is a "live-at-head" project. In other words, it is intended that the latest commit on the 'master' branch will always be the best version to develop with. To aquire it, simply clone it from *GitHub*:

```
git clone https://github.com/gbpoole/gbpSID.git
```

However, if you want to download the latest version specifically tagged as a standardized release, try this:

```
git clone --branch "`git ls-remote --tags https://github.com/gbpoole/gbpSID.git | sed
↪'s|.*/\(.*\)$|\1|' | grep -v '\^' | sort -t. -k1,1nr -k2,2nr -k3,3nr`" https://
↪github.com/gbpoole/gbpSID.git
```

1.2 Configuring the library

Once cloned, create a build directory (for example):

```
cd gbpSID
mkdir build
```

Then move to that directory and run `cmake` on the project directory (i.e. the directory where the file *CMakeLists.txt* is located):

```
cd build
cmake ..
```

Several options can be passed to `cmake` to tailor *gbpSID* to your needs. These are as follows:

ADD TABLE HERE.

1.3 Building the library

Once configured with `cmake`, the project can be built by moving to the build directory and running the following:

```
make
```

To install the project, specify the installation directory as follows:

```
make DESTDIR=<full-path-to_installation-location> install
```

1.4 Building the documentation

Once `cmake` has been run, documentation can be built by running the following from the build directory:

```
make docs
```

This will place a *.pdf* version of the documentation in the directory *docs* and an *html* version in *docs/html/docs*.

1.5 Installing as a submodule

ADD TEXT HERE.

CHAPTER 2

Applications

2.1 The Application Class

2.2 gbpSID_hello_world

usage:

```
gbpSID_hello_world <sender> options
```

where options are:

-e, –enthusiastic add enthusiasm to the message

-?, -h, –help display usage information

This program demonstrates how to configure and use an application program with the gbpApp class.

CHAPTER 3

C/C++ API

3.1 Core

3.1.1 Class definitions

struct SID_info

Public Members

SID_Comm ***COMM_WORLD**
The communicator used by SID.

FILE ***fp_log**
File pointer for log output.

int **My_rank**
Rank of process in SID_COMM_WORLD.

int **n_proc**
Number of processes in SID_COMM_WORLD communicator.

int **logging_active**
Used to control whether a rank can write to the log or not.

int **verbosity**
Sets the maximum indentation level of the log.

int **level**
Sets the current indentation level of the log.

int **indent**
If evluates to True, then next log entry needs to be indented.

int **I_am_Master**
SID rank identified as the “Master Rank”.

int **I_am_last_rank**
SID rank which is identified as the "Last Rank".

int **rank_to_left**
SID rank identified as being to the "left" of the current rank.

int **rank_to_right**
SID rank identified as being to the "right" of the current rank.

time_t **time_start**
Time of application execution start.

time_t **time_stop**
Time of application execution end.

time_t ***time_start_level**
Time of start for all active indentation levels.

time_t ***time_stop_level**
Time of end for all active indentation levels.

double ***IO_size**
I/O Size for I/O progress counter.

int ***time_total_level**
Total time spent in this indent level.

int ***flag_use_timer**
True if timing reporting is to be reported for this indent level.

char ***My_node**
Name of the rank's node.

char **My_binary**[SID_MAX_FILENAME_LENGTH]
Application executable name.

struct gbp_va_list
#include <*gbpSID_core.h*> Variadic arguments structure.

Public Members

char **stream**[SID_MAX_VARGS_STREAM_SIZE]
A c-style string specifying the argument.

int **stream_position**
Position in the list.

3.1.2 Functions

void **SID_Init** (int **argc*, char ***argv*[], void **mpi_comm_as_void*)
Initialize the SID run-time environment This function should be called as soon as possible for any project utilizing *gbpSID*. It takes pointers to the run-time arguments passed to main() and an optional communicator to inherit from as parameters

Parameters

- `argc`: A pointer to the argument count passed to main()
- `argv`: A pointer to the argument list passed to main()
- `mpi_comm_as_void`: An optional MPI communicator to inherit from. Set to NULL to ignore.

void **SID_Finalize**()
Clean-up a SID runtime configuration.

void **SID_exit_error**(**const** char *_fmt_, int _r_val_, ...)
Exit with an error

Parameters

- fmt:
- r_val:
- ...:

void **SID_seconds2ascii**(int _n_secs_, char *_string_)
Convert a number of seconds to a c-style string describing the amount of time represented by it

Parameters

- n_secs:
- string:

void **SID_va_start**(_gbp_va_list_ *_vargs_)
Initialize a variadic argument list

Parameters

- vargs:

void **SID_add_va_arg**(_gbp_va_list_ *_vargs_, size_t _size_, void *_ptr_)
Add a variadic argument

Parameters

- vargs:
- size:
- ptr:

void **SID_fetch_va_arg**(_gbp_va_list_ *_vargs_, size_t _size_, void *_ptr_)
Return the content of a variadic argument

Parameters

- vargs:
- size:
- ptr:

3.1.3 Error code definitions

SID_ERROR_NONE
No error.

SID_ERROR_LOGIC
Generic error in logic.

SID_ERROR_IO_OPEN
I/O open error.

SID_ERROR_IO_READ
I/O read error.

SID_ERROR_IO_WRITE
I/O write error.

SID_ERROR_MEMORY
Memory allocation error.

SID_ERROR_SYNTAX
Syntax error

3.2 Memory managment

3.2.1 Functions

void ***SID_malloc**(size_t *allocation_size*)

void ***SID_calloc**(size_t *allocation_size*)

void ***SID_realloc**(void **original_pointer*, size_t *allocation_size*)

void **SID_free**(void ***ptr*)

3.3 Logging

3.3.1 Functions

void **SID_log**(**const** char **fmt*, int *mode*, ...)

void **SID_log_error**(**const** char **fmt*, ...)

void **SID_log_set_fp**(FILE **fp*)

void **SID_log_warning**(**const** char **fmt*, int *mode*, ...)

void **SID_log_header**()

void **SID_log_footer**()

void **SID_set_verbosity**(int *mode*, ...)

void **SID_Init_pcounter**(pcounter_info **pcounter*, size_t *n_i*, int *n_report*)

void **SID_check_pcounter**(pcounter_info **pcounter*, size_t *i*)

3.4 Message Passing Interface (MPI)

3.4.1 Functions

void **SID_Comm_init**(SID_Comm ***comm*)

void **SID_Comm_free**(SID_Comm ***comm*)

void **SID_Comm_list**(SID_Comm **comm_in*, int *comm_id*, SID_Comm **comm_out*)

void **SID_Comm_split**(SID_Comm **comm_in*, int *colour*, int *key*, SID_Comm **comm_out*)

void **SID_Type_size**(SID_Datatype *type*, int **size*)

void **SID_Send**(void **sendbuf*, int *sendcount*, SID_Datatype *sendtype*, int *dest*, int *sendtag*, SID_Comm **comm*)

void **SID_Isend**(void **sendbuf*, int *sendcount*, SID_Datatype *sendtype*, int *dest*, int *sendtag*, SID_Comm **comm*, SID_Request **request*)

void **SID_Ssend**(void **buf*, int *count*, SID_Datatype *datatype*, int *dest*, int *tag*, SID_Comm **comm*)

void **SID_Recv**(void **recvbuf*, int *recvcount*, SID_Datatype *recvtype*, int *source*, int *recvtag*, SID_Comm **comm*, SID_Status **status*)

void **SID_Irecv**(void **recvbuf*, int *recvcount*, SID_Datatype *recvtype*, int *source*, int *recvtag*, SID_Comm **comm*, SID_Request **request*)

void **SID_Sendrecv**(void **sendbuf*, int *sendcount*, SID_Datatype *sendtype*, int *dest*, int *sendtag*, void **recvbuf*, int *recvcount*, SID_Datatype *recvtype*, int *source*, int *recvtag*, SID_Comm **comm*, SID_Status **status*)

void **SID_Probe**(int *source*, int *tag*, SID_Comm **comm*, SID_Status **status*)

void **SID_Reduce**(void **sendbuf*, void **recvbuf*, int *count*, SID_Datatype *datatype*, SID_Op *op*, int *root*, SID_Comm **comm*)

void **SID_Allreduce**(void **sendbuf*, void **recvbuf*, int *count*, SID_Datatype *datatype*, SID_Op *op*, SID_Comm **comm*)

void **SID_Allgather**(void **sendbuf*, int *sendcount*, SID_Datatype *sendtype*, void **recvbuf*, int *recvcount*, SID_Datatype *recvtype*, SID_Comm **comm*)

void **SID_Allgatherv**(void **sendbuf*, int *sendcount*, SID_Datatype *sendtype*, void **recvbuf*, int **recvcounts*, int **displs*, SID_Datatype *recvtype*, SID_Comm **comm*)

void **SID_Gatherv**(void **sendbuf*, int *sendcount*, SID_Datatype *sendtype*, void **recvbuf*, int **recvcounts*, int **displs*, SID_Datatype *recvtype*, int *root*, SID_Comm **comm*)

void **SID_Scatterv**(void **sendbuf*, int **sendcounts*, int **displs*, SID_Datatype *sendtype*, void **recvbuf*, int *recvcount*, SID_Datatype *recvtype*, int *root*, SID_Comm **comm*)

void **SID_Barrier**(SID_Comm **comm*)

void **SID_Bcast**(void **buffer*, int *count*, SID_Datatype *datatype*, int *source_rank*, SID_Comm **comm*)

void **SID_Waitall**(int *count*, SID_Request *array_request*[], SID_Status *array_status*[])

double **SID_Wtime**(void)

3.5 File I/O

3.5.1 Functions

int **SID_fopen**(**const** char **filename*, **const** char **mode*, SID_fp **fp*)
Open a SID file pointer

Return

Parameters

- `filename`:
- `mode`:
- `fp`:

int **SID_fclose**(SID_fp **fp*)
Close a SID file pointer

Return

Parameters

- fp:

size_t **SID_fread**(void *_buffer_, size_t _size_per_item_, size_t _n_items_, SID_fp *_fp_)
Perform a rank-independent read with a SID file pointer

Return

Parameters

- buffer:
- size_per_item:
- n_items:
- fp:

size_t **SID_fwrite**(void *_buffer_, size_t _size_per_item_, size_t _n_items_, SID_fp *_fp_)

void **SID_frewind**(SID_fp *_fp_)

void **SID_fseek**(SID_fp *_fp_, size_t _size_per_item_, size_t _n_items_, int _origin_)

void **SID_fskip**(size_t _size_per_item_, size_t _n_items_, SID_fp *_fp_)

void **SID_fseek_end**(SID_fp *_fp_)

size_t **SID_fread_all**(void *_buffer_, size_t _size_per_item_, size_t _n_items_, SID_fp *_fp_)
Perform an identical read to all ranks with a SID file pointer

Return

Parameters

- buffer:
- size_per_item:
- n_items:
- fp:

size_t **SID_fwrite_all**(void *_buffer_, size_t _size_per_item_, size_t _n_items_, SID_fp *_fp_)

void **SID_fread_all_buffer**(void *_rval_, size_t _dtype_size_, size_t _n_items_, SID_fp_buffer *_fp_buffer_)

size_t **SID_fread_ordered**(void *_buffer_, size_t _size_per_item_, size_t _n_items_, SID_fp *_fp_)

size_t **SID_fwrite_ordered**(void *_buffer_, size_t _size_per_item_, size_t _n_items_, SID_fp *_fp_)

size_t **SID_fwrite_shared**(void *_buffer_, size_t _size_per_item_, size_t _n_items_, SID_fp *_fp_)

int **SID_fopen_chunked**(**const** char *_filename_root_, **const** char *_mode_, SID_fp *_fp_, void *_header_, ...)
Open a chuncked SID file pointer

Return

Parameters

- filename_root:
- mode:
- fp:
- header:

- ...:

int **SID_fclose_chunked**(SID_fp *fp)
Close a chuncked SID file pointer

Return

Parameters

- fp:

size_t **SID_fread_chunked**(void *buffer, size_t n_x_read_local, size_t i_x_offset_local, SID_fp *fp)

size_t **SID_fread_chunked_all**(void *buffer, size_t n_x_read, SID_fp *fp)

size_t **SID_fread_chunked_ordered**(void *buffer, size_t n_x_read_local, SID_fp *fp)

void **SID_frewind_chunked**(SID_fp *fp)

void **SID_fseek_chunked**(size_t i_x_skip_local, SID_fp *fp)

void **SID_fskip_chunked**(size_t n_x_skip_local, SID_fp *fp)

size_t **SID_fwrite_chunked**(void *buffer, size_t n_x_write_local, size_t i_x_offset_local, SID_fp *fp)

int **SID_remove_chunked**(char *filename_root)

void **SID_cat_files**(**const** char *filename_out, int mode, int n_files, ...)
Concatinate a set of files

Parameters

- filename_out:
- mode:
- n_files:
- ...:

void **SID_Init_fp_buffer**(SID_fp *fp, size_t n_bytes_to_read, size_t n_bytes_buffer_max, SID_fp_buffer **fp_buffer)

void **SID_reset_fp_buffer**(SID_fp_buffer **fp_buffer)

void **SID_free_fp_buffer**(SID_fp_buffer **fp_buffer)

size_t **SID_fread_verify**(void *ptr, size_t size, size_t count, FILE *stream)

3.6 Development tools

3.6.1 Functions

void **SID_mpi_gdb_here**()
Set a gdb breakpoint.

void **SID_test**(int val, **const** char *fmt, ...)
Output a debug test message

Parameters

- val:
- fmt:
- ...:

- genindex

Index

G

S