Adaptive Impact-Driven Detector (AID) User Guide

(Version 0.2)

Mathematics and Computer Science (MCS)
Argonne National Laboratory
Contact: Sheng Di (sdi1@anl.gov)

April 21th, 2015 (updated on Feb. 2019)

Table of Contents

1. Brief description

- AID provides a way for HPC users of dynamic simulations over multiple time steps to detect corruptions that impact the results of their execution.
- AID is designed to monitor the state data of the application: variables that are the outcome of the execution.
- AID is a library offering functions to help programmers defining which variable should be monitored.
- AID offers only detection. For recovery we suggest to combine AID with FTI. But AID could be used in combination with any other recovery library.
- AID is simple to use.
- AID works with very low overhead (including low memory cost, computation cost and communication overhead)
- AID supports both C and Fortran.

- Note: In order to support adaptive version (our TPDS paper: Adaptive Impact-Driven Detection of Silent Data Corruption for HPC Applications), you need to use **SDC_Increment_FPNum**().

2. How to download and install AID

The AID software can be downloaded from http://collab.mcs.anl.gov/display/ESR/AID

Follow the following four steps to make the installation:

a) Modify the Makefile under [AID_INSTALL_PATH], as follows:

In the Makefile, replace [AID_INSTALL_PATH] by your AID path, and replace [MPI_INSTALL_PATH] by your MPI installation path.

For example,

```
AIDPATH          = /home/shdi/aid-0.2
MPIPATH          = /home/shdi/mpich-install
```

b) Set AID PATH as environment variable (such as in ~/.bashrc):

```
export SDCHOME=[AIDPATH]/SDC
```

#You need to replace [AIDPATH] by your AID installation path, such as /home/shdi/aid-0.1

c) Set LD_LIBRARY_PATH as follows:

```
export LD_LIBRARY_PATH= $SDCHOME/lib:$LD_LIBRARY_PATH
```

d) Go to the [AIDPATH], run the following commands:

make

make install

Note:

(1) If your MPI program is coded in Fortran, you don't have to modify Makefile any more. If your MPI application is coded in C, you need to further comment out the following line in the Makefile.

```
# $(OBJ)/sdcf.o $(OBJ)/sdc_interface.o $(OBJ)/fort_writefile.o \
```

(The above line is only customized for Fortran version)

(2) Recompile the whole library: make clean;make;make install

3. How to protect MPI programs

There are only four steps for users to annotate their MPI application codes:

(1) Initialize the detector by calling SDC_Init();

(2) Specify the key variables to protect by calling SDC_Protect(var,ierr);

(3) Annotate the execution iterations by inserting SDC_Snapshot() into the key loop;

(4) Release the memory by calling SDC_Finalize() in the end.

In what follows, we describe the key functions/interfaces as well as the parameters, for both C interface and Fortran interface, respectively. The mandatory interfaces are SDC_Init, SDC_Protect, SDC_Snapshot, and SDC_Finalize. Other interfaces are optional, based on user's demand. For example, the last three interfaces (Cost_Start(), Cost_End() and Print_Cost_Ratio()) are used to collect the detection overhead of the detector.

(a) SDC_Init

Initialize the SDC detector.

Synopsis:

C: int SDC_Init(char *configFile, MPI_Comm globalComm);

Fortran: SDC_Init(CHARACTER(LEN=*) configFile, int ERR)

Input Parameters:

configFile	configuration file
globalComm	global communicator, such as MPI_COMM_WORLD

Usage:

- SDC_Init is often called in the beginning of the MPI program, e.g., right after the MPI_Init(), MPI_Comm_Size() and MPI_Comm_Rank(). (see examples/heatdis.c)
- We also provide a specific initialization handler called SDC_Init_nonMPI, which is used only for debugging. It doesn't require configuration file and communicator as the input parameters. (see examples/simple_c_4d.c)

(b) SDC_Protect

Specify the key variables to be protected by the detector. The current version is able to protect the array variables with maximum 5 dimensions.

Synopsis:

C:

SDC_Protect(char* var_name, void* data, int data_type, int r5, int r4, int r3, int r2, int r1)

Fortran: SDC_Protect(CHARACTER(LEN=*) var_name, void* data)

Input Parameters:

var_name	variable name given by users
data	the variable to be monitored in the execution
data_type	Type of data (only three options: SDC_INTEGER, SDC_FLOAT or SDC_DOUBLE)
r5	size of dimension 5
r4	size of dimension 4
r3	size of dimension 3
r2	size of dimension 2
r1	size of dimension 1

Usage:

- data_type is used to specify the type of the variable: integer, float or double.

For C: the user needs to specify the data_type to one of the three options.
For Fortran: the user just need to give two parameters as shown above.

- The dimension of the variable is determined based on the five dimension parameters (r5, r4, r3, r2, and r1). For instance, if the variable is a 2D array (MXN), then r5=0, r4=0, r3=0, r2=M, and r1=N. If the variable to protect is a 4D array, then only r5 is set to 0. (See simple_c_4d.c for details)

(c) SDC_Snapshot

Annotate the execution iterations

Synopsis:

C: SDC_Snapshot()

Fortran: SDC_Snapshot(CHARACTER(LEN=*) var_name, void* data, INTEGER ERR)

Input Parameters:

var_name	variable name given by users
data	the variable to be monitored in the execution
ERR	the detection result

Usage:

- For C version, the variables have been registered by SDC_Protect(), so SDC_Snapshot() doesn't require users to pass them again.
- For Fortran, users need to pass the variable again, because it is implemented by the fortran-iso-C-binding, under which the addresses of the variables vary when they are passed from Fortran to C each time.
- Obviously, Fortran users need to call multiple SDC_Snapshot functions in the key loop if there are multiple variables to protect. (Each variable has a SDC_Snapshot call in the key loop of the program).
- The detection result at this iteration will be returned as a return_value for C, and will be passed out through the parameter ERR for Fortran.

  The detection result = 0 (no SDC) or 1 (SDC exists in this iteration)

(d) SDC_Increment_FPNum

Increase the number of false positive iterations during the simulation, in order to adaptively tune the error detection range (or radius). For the details, please see the equation (6) in our TPDS paper.

Synopsis:

C: SDC_Increment_FPNum()

Fortran: SDC_Increment_FPNum()

Usage:

Whenever the program detects an error based on SDC_Snapshot(), the program needs to fix the error by rerunning the previous 'checkpointed' iteration. If the error happened again, the program should (likely) think of this error as a false positive. Then, the code needs to use SDC_Increment_FPNum() to mark it. Some example codes are shown below:

```
for (;;)
{
        doWork….
```

```
                sdc_result=SDC_Snapshot(); /////////////detector
                SDC_SetMark(i,sdc_result); /////////////mark error
                if(sdc_result!=0)
                            SDC_Increment_FPNum(); //increment false-positive count
     }
```

(e) SDC_Increase_Counter

Increase the counter of the detector, making it consistent with the time steps

Synopsis:

C: No need

Fortran: SDC_Increase_Counter()

Input Parameters: none.

Usage:

- For C version, the user doesn't have to use this function.
- For Fortran, the user needs to put it somewhere in the key loop (such as in the end of the key loop).

(f) SDC_SetMark

Store the detection results, to be printed by SDC_PrintDetectResult() in the end.

Synopsis:

C: int SDC_SetMark(int time_step, int result)

Fortran: SDC_SetMark(INTEGER time_step, INTEGER result, INTEGER IERR)

Input Parameters:

time_step	the time step of the loop (i.e., iteration number)
result	the detection result

(g) SDC_Finalize

Finalize the detector by releasing the memory.

Synopsis:

C: void SDC_Finalize()

Fortran: void SDC_Finalize()

Input Parameters:

Usage:

- SDC_Finalize is often called after MPI_Finalize()

(h) SDC_PrintDetectResult

Print the detection results

Output format: 0:X 1:X 2:X 3:X 4:X 5:X ……, where 0,1,2,… refer to the iteration numbers, and X indicates the results (either 0 or 1).

Synopsis:

C: void SDC_PrintDetectResult()

Fortran void SDC_PrintDetectResult()

Input Parameters:

Usage:

- SDC_PrintDetectResult is often called after the key loop.

(i) Cost_Start

Checking the time cost of operations, such as the cost of Snapshot().

to specify the start point of the timer.

Synopsis:

C: void Cost_Start()

Fortran void Cost_Start()

Input Parameters:

(j) Cost_End

to specify the end point of the timer.

Synopsis:

C: void Cost_End()

Fortran void Cost_End()

Input Parameters:

(k) Print_Cost_Ratio

Print the total cost accumulated by using Cost_Start() and Cost_End()

Synopsis:

C: void Print_Cost_Ratio()

Fortran: void Print_Cost_Ratio()

Input Parameters:

Usage:

- The interface Print_Cost_Ratio() should be called after SDC_Finalize()

4. Preliminary configuration

The configuration file is 'config.sdc", which can be found in the [AIDPATH]/examples

Parameter name	**Description**
sol_name	AID, LCF or ABF AID: The recommended solution (Adaptive Impact-Driven Detector) LCF: Linear Curve Fitting ABF: Alpha-beta Filter, which is identical to Quadratic Curve Fitting
samp_distance	sample_distance is used by both AID nad non-AID for different purposes. For AID: samp_distance is used for determining the number of samples used to search bestfit orders. If samp_distance is set to 3, then the number of sample data for determining the bestfit prediction methods periodically is one third of the total number of data points per snapshot. For non-AID: samp_distance is valid only when sampling >= 1
impact_err_bound_ratio	impact_err_bound_ratio is set to 0.00078125 for FLASH, and 0.05 for heatdis.c. For details, please read our paper [1].

Lambda	the coefficient to determine the outstanding solution set, see [1] for details.
check_bestfit_rate	how often to search the bestfit order, e.g., do it every 20 iterations
fixed_value_range	If the global value range is fixed, users can set its value here If there are many variables to protect, fixed_value_range should be set to the minimum value. fixed_value_range = -1 means to use the dynamic value range. fixed_value_range != -1 means to use static value range and the range value is always equal to the number assigned to fixed_value_range. fixed_value_range != -1 can avoid the communication cost of MPI_Allreduce(max,min)
collect_value_range_rate	how often to compute the global value range
Floor	floor: the upper bound when the threshold is going to be set to 0 (i.e., to avoid the threshold=0) #as for AID: for example, when the interval of global_value_range =0, then the detection range will be [X-0,X+0]. the floor will be used to avoid this situation.

5. Test-cases (examples)

HeatDistribution:
Compile: Get in the directory [AIDPATH]/examples, and execute "make hd"
[source code: heatdis.c]
Run: make hdt

Hint: If you want to output the results of the HeatDistribution, you need to add -DINTERACTIVE when you compile it. Modify [AIDPATH]/examples/Makefile.

simple_fortran_1d:
Get in the [AIDPATH]/examples, and execute "make simple_fortran_1d"
[source code: simple_fortran_1d.f90]
Run: make test_fortran_1d

6. Trouble shooting

a) I cannot compile C mpi program with AID library, with the following errors:

```
[fti@localhost examples]$ make hd
mpicc -o hd heatdis.c -I/home/shdi/aid-0.1/SDC/include -L/home/shdi/aid-0.1/SDC/lib
-lsdc -lm #-DINTERACTIVE
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_st_close'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to
`_gfortran_transfer_character_write'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to
`_gfortran_transfer_integer_write'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_internal_unpack'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_st_write_done'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to
`_gfortran_transfer_real_write'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_system_sub'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_size0'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_access_func'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_reshape_r8'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_st_write'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_internal_pack'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_st_open'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_string_trim'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_concat_string'
/home/shdi/aid-0.1/SDC/lib/libsdc.so: undefined reference to `_gfortran_string_len_trim'
```

Reason: You compiled the AID in the Fortran mode, and then you are compiling a MPI program coded in C.
Solution: Modify [AID_INSTALL_PATH]/Makefile by commenting out the following line:
$(OBJ)/sdcf.o $(OBJ)/sdc_interface.o $(OBJ)/fort_writefile.o \
And then, recompile the AID library by running "make clean;make;make install"

b) I cannot compile the example programs in the examples/. The errors prompted are as follows:

```
[fti@localhost examples]$ make simple_c_4d
mpicc -g -o simple_c_4d simple_c_4d.c -I/home/fti/aid-0./SDC/include
-L/home/fti/aid-0./SDC/lib -lsdc -lm
simple_c_4d.c:3:26: fatal error: sdc_detector.h: No such file or directory
 #include "sdc_detector.h"
                          ^
compilation terminated.
make: *** [simple_c_4d] Error 1
```

Reason: You didn't set the correct SDCPATH in [AID_INSTALL_PATH]/examples/Makefile.
Solution: Set SDCPATH in the [AID_INSTALL_PATH]/examples/Makefile.

c) I can compile the example programs, but cannot run them, with "undefined symbol" errors.

Reason: The environment variable LD_LIBRARY_PATH is set incorrectly.

Solution:

Set LD_LIBRARY_PATH as follows:

```
export LD_LIBRARY_PATH= $SDCHOME/lib:$LD_LIBRARY_PATH
```

<END>