Synchronizing the Record-Replay engines for MPI Trace compression

By
Apoorva Kulkarni
Vivek Thakkar

PROBLEM DESCRIPTION

An analysis tool has been developed which characterizes the communication behavior of large scale massively parallel applications which use MPI to communicate amongst parallel tasks. This tool has essentially two components – record engine and replay engine. Record engine efficiently produces the communication traces in compressed file(s) and replay engine replays the execution and generates statistical data for analysis purpose. The current record engine has some extensions done over a period of time. However, the corresponding changes have not been done to the replay engine. So, our task is to identify the changes that are needed and to make those changes and do the benchmarking with small and large benchmarks.

Solved Issues

1. Fixing the new record engine.

We have managed to successfully generate the rsd files for the IS benchmark in the NAS Parallel Benchmarking suite. This required three fixes in the code.

First fix:

While trying to run the benchmark with the new record framework we found that most of the calls were successfully being captured by the record engine but the MPI_Alltoallv() call was aborting in the middle of execution. On further analysis of the code we found that initialization of the value for sender's displacement was incorrect. The fix was nothing but a change in the name of the variable used to denote sender's displacement in the mpi-spec.umpi.extract file. Instead of being “sdispl” it was named “displ” in the file.
Changed File: mpi-spec.umpi.extract

Second Fix:

Secondly, the communicator size does not seem to be retrieved in the data structure “op-> data.mpi.size”. This always returns 0 and the allocation of the amount of memory for parameters like sender_count, receiver_count, sender_displacement and receiver_displacement in MPI_Alltoallv does not happen and therefore replay fails.
Changed File: mpi-spec.umpi.extract

Third Fix:

Here we used the stack_sig.h file that was modified by Prasun Ratn earlier. The record framework generates a unique call site address by using frame pointers stored in an execution environment variable by setjmp system call. Since, the code was originally written for blue gene machine with different architecture and different registers, the method of generating the unique sequence had to be modified for X86 architecture. However, the limitation of this fix is that we need to turn off the optimisation flags in the following files :
For record Engine: config/Makefile.config

For replay Engine: Makefile
For Test mpi program: Disable any optimisation flag.
This is because with optimization enabled compiler does an inlining of call frames. For further clarification, please refer to the message board.
Changed File: stack_sig.h

2. Implementation of MPI_Alltoallv() in replay engine

The MPI_Alltoallv() function was not being implemented initially in the old replay framework. We have added the implementation code to the prsd_utils.c (it is inverse of record's implementation) in the replay source. But we have not been able to successfully perform the replay operation for that call and hence for the IS benchmark. However the printings illustrate that the implementation is successful but we are currently debugging the replay side of the benchmark which is almost complete (One thread reaches at the final stage and is waiting for other threads. It seems that there are some issues with asynchronous mpi operations).
We were able to locate the problem. After a lot of effort it was observed that record does not write the src/destination information in the rsd files if they are at an offset of -1. And the replay was not initializing the structure elements which copy these elements. And this caused the inconsistency in the output results. We have not fixed this by initializing them with -1, the default src/destination.
Further running the benchmark, we found that our MPI_Alltoallv() was giving an error: "Truncated Length". We had to debug a lot for that. We ran small programs with MPI_Alltoallv and they gave success. Finally the problem has been isolated and it looks that compression in record does not correctly do parameter matching. As a result It would probably not be possible to fix this till the time of submission of report.

3. Handling offsets in replay

Record generates source and destination identifiers in Recv/Irecv and Send/Isend in terms of offsets from the current rank. However, replay has been implemented to take absolute identifiers i.e. the rank of the node as the send/receive identifier. Hence, we have calculated the absolute value of the ids by adding the rank in offsets. Fix has been done in the function lookup_prsd_call(...) in the handling of MPI_Send,MPI_Isend,MPI_Recv and MPI_Irecv.
Changed File: prsd_utils.c

4. Correct Handling of Offset for Request handle

On debugging we found that errors like MPI_Isend: Unhandled Parameter were coming. We analyzed this problem and found out that offset was incorrectly initialized in handle for MPI_Wait. We have fixed this and tested it and it now works.
Changed File: prsd_utils.c

5. A small Hack

If replay does not recognize some prsd_content in the generated rsd file, it exits leaving other threads

waiting. To prevent this from hampering the outcome of the result, we have commented exit(0) call. For example, the new record generates a printing of “all <execution time> “. This is not recognized by our replay. With our hack it is able to run successfully but just prints out “unhandled parameter” . This is not a fatal bug and if time permits, we may try to resolve this later on at lower priority.

6. Linking mpiP with the old replay engine

The old replay engine was successfully linked with the mpiP profiling framework. So if the rsd files were properly generated by the record engine and then the replay engine should be able to perform its job and the mpiP successfully creates a profile of the execution.
Changed File: replay/Makefile

Unsolved Issues

1. Although it appears that the record engine has been fixed for some common problems a slew of tests need to be performed to ensure that the new record engine is working fine and proper rsd output files are generated. So far tests have been performed using some of the files provided in the tests folder of the record source. We were able to execute some of them successfully. As mentioned earlier we were also able to execute the IS benchmark using the record engine. But more benchmarking needs to be performed to test that most of the general MPI_foo() calls are captured.

2. Some MPI operations seemingly unimportant in terms of their use in benchmarks have been left unimplemented. We need to decide if we can implement a few of them within the time constraint. Some of the operations which we may need to implement could be MPI_Scatter , MPI_Scatterv, MPI_Abort. But we need to reach a final agreement on this with our instructor.

CONTRIBUTIONS BY MEMBERS

APOORVA KULKARNI - was instrumental in compiling both the record and replay engines and doing the major modifications needed in the Makefile, also performed testing with the modified replay engine and is maintaining our web page.

VIVEK THAKKAR - tested and debugged the old replay engine and made some useful modifications in files prsd_utils.c and request_handler.c to get the framework in running state, also has made this report.

Effort has been put by both team members to fix the new record engine which required huge amount of debugging.

Test Cases Successfully done

Following MPI programs from the test programs have been tested on 2-4 nodes:

Hello.exe
Send-Recv-Ok.exe
Send-Recv-Ok2.exe
Irecv-Isend-Ok.exe
Irecv-Isend-Ok2.exe

Our own test examples:
MPI program to calculate PI: mpipi.exe
MPI_Alltoallv() example: alltoallv.exe
Program to test MPI_AllReduce(): allredex.exe

Following MPI calls were covered in these tests:

MPI_Init()
MPI_Send()
MPI_Recv()
MPI_Irecv()
MPI_Isend()
MPI_Alltoall()
MPI_Barrier()
MPI_Bcast()
MPI_Reduce()
MPI_Allreduce()
MPI_Alltoallv()
MPI_Finalize()

Test Cases Planned

- Scaling replay for IS to 16 nodes: Record has been tested successfully with 16 nodes.
- Slew of other test cases with specific MPI calls.
- Matrix Multiplication from HW1 and HW2

Conclusion

Following action points were planned at the start of project:

1. Take the source code for old record engine and old replay engine (given in src.tar.gz).

2. Compile record and replay. Make changes in config/Makefile.config, libsrc/stack_sig.h and libsrc/util.c by comparing with the same files given in the new record. Make other changes if required.

3. Try to run sample programs given in “tests” subdirectory. First run by linking them with libumpire.a which is generated as a result of compiling the record engine. Do the initial runs on 1 processor. The

record engine should produce the desired trace file(s). Now run the replay engine (linked with mpiP) giving it the path of the trace files Verify the outputs against outputs produced by mpiP. Test 2-3 sample programs. Do the runs on 2, 3 and 4 processors. Perform testing on at least one large benchmark like NAS.

4. Completion of step 3 ensures that old record and replay are synchronously working. Now, make this old replay file as the base file on which the extensions need to be merged. Doing a diff on the libsrc sub directories of the old record and new record shows some minor differences in the following files: Makefile, mpi-spec.umpi.extract, rsd_queue.h, umpi_internal.c, umpi_internal.h,,transfer.h, umpi_mpi.c, umpi_mpi_lookup.c. The action point would be to understand the changes and their motivation. Also an intermediate report (HW5) has to be submitted.

5. Establish a common understanding to identify the changes that may be needed in the replay engine corresponding to the changes done in the record engine. Make those changes on the base replay file (refer 4) to get the new replay engine in sync with the new record engine. They are anticipated to be very few (nonetheless very essential).

6. Compile the code and run small benchmarks as done in step 3. Run the tests on large benchmarks like NAS,ASCI, purple etc. Use different DEBUG switches ref. README of record engine) to ensure the correctness of intermediate and final outputs. While benchmarking try to find if some changes can be made in the algorithm to ensure better compression.

We have managed to cover 1,2,3 fully. For action point 4, we have compared the differences of old record and new record and have reached the conclusion that nothing needs to be added on replay side. The only major difference we found in new record was the implementation of timing statistic at the end of the execution. However, for replay side, timing delta implementation is a separate project handled by a different team. So, we can say that point 4 has been covered.
Point # 5 is a bit unclear to us at the moment since we need to know which MPI calls should we implement in replay which could cover most of the benchmarks. Action point 6 depends on action point 5.

REFERENCES

[1] Class Slides and Paper by M. Noeth, F. Mueller, M. Schulz, B. de Supinski, Scalable Compression and Replay of Communication Traces in Massively Parallel Envrionments, submitted to IPDPS 2007

[2] Dynamic Software Testing of MPI Applications with Umpire
www.llnl.gov/CASC/people/vetter/pubs/sc00-umpire-vetter.pdf

[3] ASCI purple benchmark
http://www.llnl.gov/asci/purple/benchmarks/limited/code_list.html

[4] NAS parallel benchmark
http://www.nas.nasa.gov/Resources/Software/npb.html

LINK TO OUR PROJECT WEBPAGE
http://www4.ncsu.edu/~askulkar/mpitrace.html

Acknowledgment

We thank the following people for their help :

-Dr. Frank Mueller for the guidance on the project.
-Prasun Ratn for the stack_sig.h and stack_sig.c fix.
-Jaydeep Marathe for helping us to figure out the bug in mpi-spec.umpi.extract.