PRUNERS

Providing Reproducibility for Uncovering Non-Deterministic Errors in Runs on Supercomputers

PMCD/JOWOG February 14\textsuperscript{th}, 2018

Dong H. Ahn (PI), Chris Chambreau, Ignacio Laguna, Gregory L. Lee, Kento Sato, Martin Schulz, Simone Atzeni, Michael Bentley, Goef Sawaya, Ganesh Gopalakrishnan, Zvonimir Rakamaric, Joachim Protze
Debugging/Testing large-scale applications is challenging.

“On average, software developers spend 50% of their programming time finding and fixing bugs.” [1]

Easy-to-fix bugs VS. Difficult-to-fix bugs

👍 Totalview/DDT

Some bugs require to use Totalview or DDT

Many bugs can be quickly fixed by simple printf debugging

👍 printf

Some class of bugs are significantly hard to fix

printf Totalview/DDT
Bugs are not created equal!

- When debugging/testing, reproducibility is very important

- Examples
  - Bugs that manifest themselves when using `-O3`, but do not with `-O0`
  - Bugs that manifests in a run and do not in the next run
Non-deterministic bugs cost substantial amounts of time and efforts

**Diablo/Hypre 2.10.1**

- The bug manifested in particular machines
- It hung only once every 30 runs after a few hours
- The scientists spent *2 months* in the period of *18 months*, and then gave up on debugging it

**HYDRA (porting on Sequoia)**

- MPI/OpenMP application non-deterministically crashed in an OpenMP region when compiling with optimization levels
- Manifested intermittently at or above 8K MPI processes
- The scientists spent *months*, and then ended up disabling OpenMP

and more ...
Non-deterministic bugs are introduced at multiple levels

- Introduced at the compiler level or at different runtime levels
- A monolithic tool won’t work for all cases
- Debugging/testing toolset
  - Individual tool works effectively
  - Interoperable and composable each other
  - Make debugging/testing easier even under other existing debuggers
The PRUNERS Toolset comprises four individual tools:

- **FLiT**: Compiler-induced floating-point computation variability tester
- **Data race detector for OpenMP programs**
- **MPI record-and-replay tool for reproducing non-deterministic MPI bugs**
- **Noise injection tool for exposing message race bugs**
Different compilers, compiler flags and platforms produce different numerical results

- No guarantee that floating-point computation on one platform is the same on another platform — E.g.) Apply associativity rules of real arithmetic

\[
\begin{align*}
\text{gcc-4.9.3 -00} \\
\text{gcc-4.9.3 -03} \\
\text{icc-16.0.3 -00}
\end{align*}
\]

- \textbf{Shewchuk/Kahan summation}

\[
\begin{align*}
float \text{ val} &= x + y; \\
float \text{ err} &= y - (\text{val} - x);
\end{align*}
\]

- \textbf{gcc-4.9.3 -03}

\[
\begin{align*}
float \text{ val} &= x + y; \\
float \text{ err} &= y - y; \\
0
\end{align*}
\]

- \textbf{icc-16.0.3 -03}

It’s important to understanding how sensitive your numerical algorithm is w.r.t. round-off errors introduced by different compilers for code verification and validation.
FLiT (Floating-point Litmus Tester)

- FLiT is a reproducibility test framework
  - Test floating-point arithmetic variability in any user-given collection of programs
- FLiT tests the variability through hundreds of combinations
  - Different compilers, compiler flags, and also different hosts
- Results are stored in SQL database and used for visualization and for further analysis
FLiT case study

- We tested several kernels which have compiler-induced FP variability
  - Difference in numerical results across different compilers, flags and kernels
- Example
  - When you want to find a compiler option that makes your applications faster while reproducing the exactly same results as non-optimized code, FLiT becomes very useful tool
OpenMP easily creates non-deterministic bugs

- Data races in OpenMP are the most malignant non-deterministic bugs
- Depending on orders of read and write, numerical results change
- Orders of read and write are non-deterministic, it introduces non-deterministic bugs

In large applications, it is difficult to manually identify data races
Archer

- Archer is a data race detector combining static and dynamic analysis

  - **Static analysis**
    - Exclude regions that can be statically detected to be race-free for dynamic analysis (Blacklisting)

  - **Dynamic analysis**
    - Detect data races based on LLVM/Clang ThreadSanitizer combined with OMPT-based annotation

Archer significantly reduce runtime overhead while providing high precision and accuracy
Archer case study: HYDRA

- Archer easily detected data races!

HYDRA (porting on Sequoia)

- MPI/OpenMP application non-deterministically crashed in an OpenMP region when compiling with optimization levels
- Manifested intermittently at or above 8K MPI processes
- The scientists spent months, and then ended up disabling OpenMP

Data races!!

```
1002: hypre_BoomerAMGInterpTruncation(...) {
    ...
1007:   int *P_diag_i = hypre_CSRMatrixI(P_diag);
1014:   int *P_offd_i = hypre_CSRMatrixI(P_offd);
1062: #pragma omp parallel private(...) 
1064:   {
    ...
1172:     if(max_elmst>0) {
    ...
1179:       for(i=start; i<stop; i++) {
1183:         last_index = P_diag_i[i+1];
1184:         last_index_offd = P_offd_i[i+1];
1188:         P_diag_i[i] = cnt_diag;
1249:         P_offd_i[i] = cnt_offd;
1484:     } /* end parallel region */
1491:   return ieer;
1492: }
```

Archer}

Data races!!

T0
Read P_diag_i[5]

T1
Write P_diag_i[5]

T0
Read P_offd_i[5]

T1
Write P_offd_i[5]
How MPI introduces non-determinism?

- It’s typically due to communication with MPI_ANY_SOURCE
- In non-deterministic applications, each MPI rank doesn’t know which other MPI ranks will send message and when

```c
Non-deterministic code w/ MPI_ANY_SOURCE

MPI_Irecv(..., MPI_ANY_SOURCE, ...);
while(1) {
    MPI_Test(flag);
    if (flag) {
        <computation>
        MPI_Irecv(..., MPI_ANY_SOURCE, ...);
    }
}
```
CORAL benchmark: MCB (Monte carlo benchmark)

- Use of MPI_ANY_SOURCE is not only source of non-determinism
  - MPI_Waitany/Waitsome/Testany/Testsome also introduce non-determinism
- MCB produces different numerical results from run to run

Example: Communications with neighbors

Non-deterministic code w/o MPI_ANY_SOURCE

```c
MPI_Irecv(..., north_rank, ..., reqs[0]);
MPI_Irecv(..., south_rank, ..., reqs[1]);
MPI_Irecv(..., west_rank, ..., reqs[2]);
MPI_Irecv(..., east_rank, ..., reqs[3]);
while(1) {
    MPI_Testsome(..., reqs, ..., flag, status);
    if (flag) {
        ...
        <computation>
        for (...) MPI_Irecv(..., status.MPI_SOURCE, ...);
        ...
    }
}
```

In non-deterministic applications,
if a bug manifests through a particular message receive order,
it’s hard to reproduce the bug, thereby, hard to debug it
ReMPI deterministically reproduce order of message receives

- ReMPI is an MPI record-and-replay tool
  - Record an order of MPI message receives
  - Replay the exactly same order of MPI message receives

- Even if a bug manifests in a particular order of message receives, ReMPI can consistently reproduce the target bug

---

SC15: Kento Sato et al., "Clock delta compression for scalable order-replay of non-deterministic parallel applications"
ReMPI case study: ParaDiS

- ParaDis
  - non-deterministically crashed after 100 to 200 iterations
- ReMPI reproduced the bug at the exactly same iteration
- ReMPI is interoperable with parallel debuggers and makes debugging non-deterministic bug easier
  - We recorded a buggy behavior in record mode
  - We diagnosed with TotalView under replay mode
ReMPI is also useful for “Testing”

- “Testing” is also important for maintaining software quality
- However, non-deterministic MPI applications present significant challenges to testing
  - The non-determinism can produce different results from run to run by nature
  - It’s difficult to reason the different numerical results are due to MPI non-determinism or software bug
- Using ReMPI, computational scientists can easily reproduce MPI behaviors, which facilitate testing

![Flowchart showing the comparison between Original code and Tuned code with and without performance tuning.](chart.png)
Unintended message races in MPI

- Many applications are written as a series of communication and computation routines executed by all processes (i.e., data parallel, SPMD)
- Developers must make sure all communication routines are “isolated”
- Example (Routine A and Routine B)
  - Different MPI_TAG or synchronization (e.g. MPI_BARRIER) between the two routines
- If not isolated, message race bugs potentially occur
  - E.g.) A message sent in Routine B is received in Routine A
- Unintended message races are non-deterministic and infrequently occur
NINJA (Noise Injection Agent Tool) exposes message race bugs by injecting noise.

NINJA detects suspicious communication routines:
- Communication routine using the same MPI_TAG without synchronization

NINJA injects a delay to MPI messages in order to enforce message races.

NINJA can test if the application has unintended message races.

PPoPP2017: Kento Sato et al., Noise Injection Techniques for Reproducing Subtle and Unintended Message Races
NINJA cast study: Diablo/Hypre-2.10.1 (in ParaSail module)

- Unintended message races in Hypre
- Prior to NINJA, the bug does not manifest itself in Hypre test code
- NINJA consistently exposed message races in the test code

Diablo/Hypre 2.10.1

- The bug manifested in particular machines
- It hung only once every 30 runs after a few hours
- The scientists spent 2 months in the period of 18 months, and then gave up on debugging it
Summary

- Debugging and testing are becoming more challenging due to non-determinism in HPC applications
- The PRUNERS toolset facilitates debugging and testing for non-deterministic applications
PRUNERS
https://pruners.github.io/

FLiT
archer
ReMPI
NINJA

Dong H. Ahn (PI)
Chris Chambreau
Ignacio Laguna
Gregory L. Lee
Kento Sato
Martin Schulz

Simone Atzeni
Michael Bentley
Goef Sawaya
Prof. Ganesh Gopalakrishnan
Prof. Zvonimir Rakamaric

Joachim Protze