PRUNERS

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

Computational Reproducibility at Exascale (CRE2017)
November 12th, 2017

Kento Sato

Lawrence Livermore National Laboratory

LLNL-PRES-741293

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Debugging/Testing large-scale applications is challenging

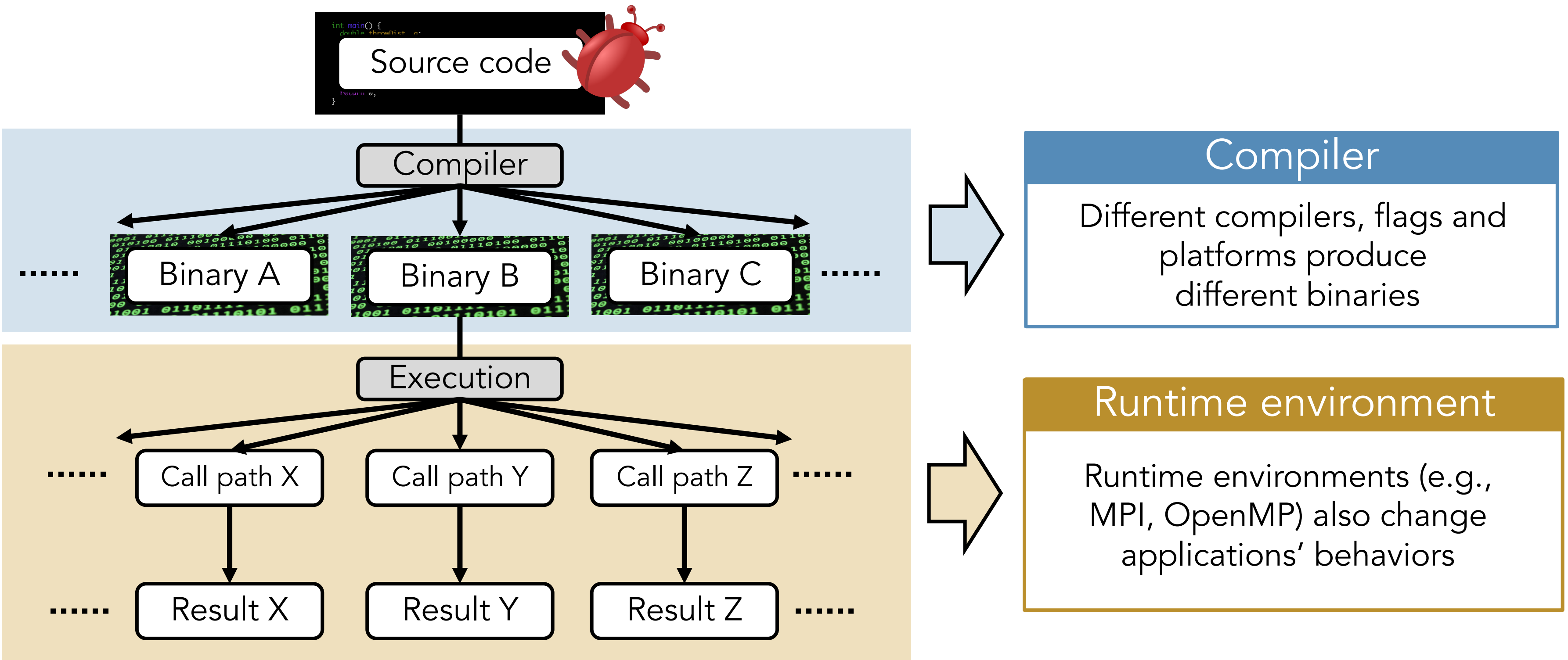
"On average, software developers spend
50% of their programming time finding and fixing bugs." [1]

Debugging/Testing are inevitable software development processes.
Tools facilitating Debugging/Testing are indispensable

[1] Source: http://www.prweb.com/releases/2013/1/prweb10298185.htm, CAMBRIDGE, UK (PRWEB) JANUARY 08, 2013

Bugs are not created equal !

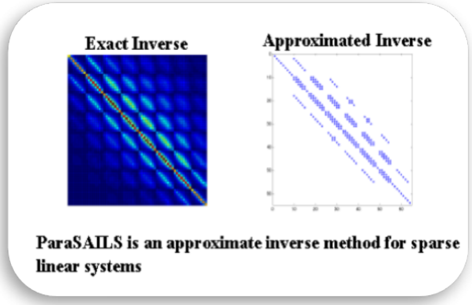
- When debugging/testing, reproducibility is very important



- Examples
  – Bugs that manifest themselves when using –O3, but do not with –O0
  – Bugs that do not manifests themselves

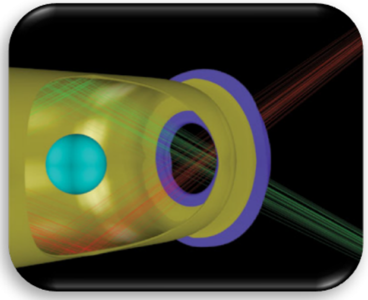
Non-deterministic bugs cost substantial amounts of time and efforts

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)

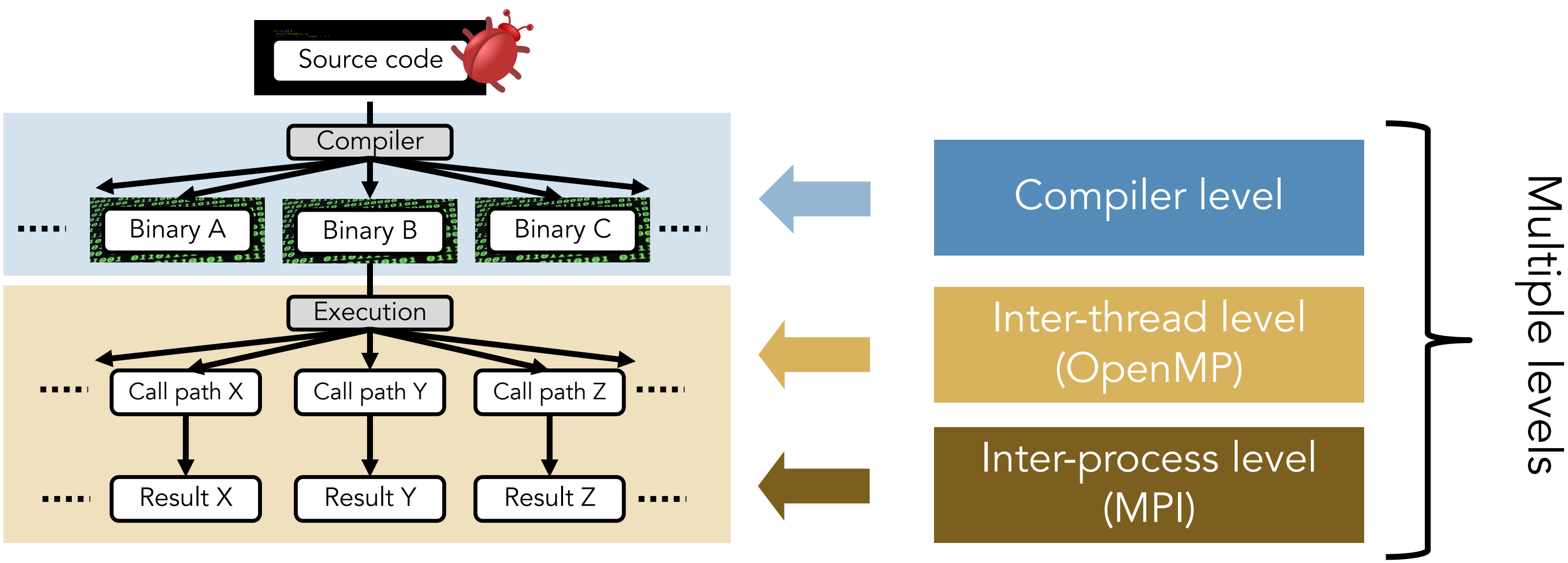


- MPI/OpenMP application non-deterministically crashed in an OpenMP region when compiling with optimization levels
- Manifested intermittently at or above 8K MPI processes
- The scientists spent months, and then ended up disabling OpenMP

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



PRUNERS Multi-level debugging/testing capabilities

The PRUNERS Toolset comprises four individual tools that can co-operate

FLiT	Compiler-induced floating-point computation variability tester	Compiler level
archer	Data race detector for OpenMP programs	OpenMP
ReMPI	MPI record-and-replay tool for reproducing non-deterministic MPI bugs	MPI
NINJA	Noise injection tool for exposing message race bugs	MPI

Multiple levels

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

```
gcc-4.9.3  -O0
gcc-4.9.3  -O3
icc-16.0.3 -O0
```

Shewchuk/Kahan summation

```
float val = x + y;
float err = y - (val - x);
```

```
icc-16.0.3 -O3
```

```
float val = x + y;
float err = y - (val - x);
```

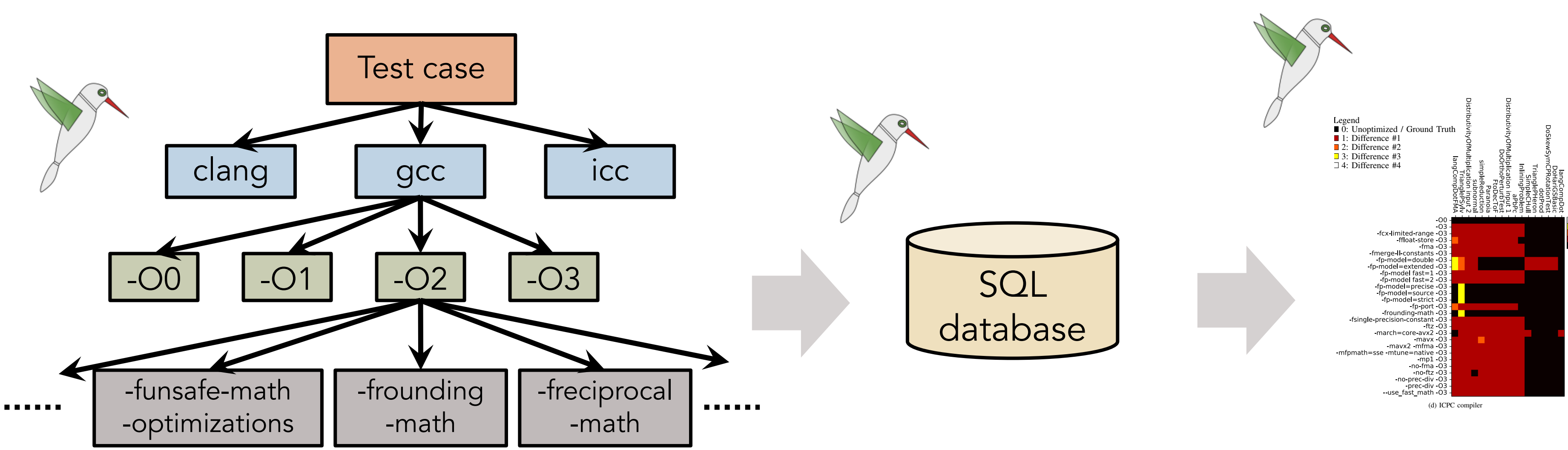
```
float val = x + y;
float err = y - y;
```

0

Understanding how sensitive your algorithm is w.r.t. to different round-off errors introduced by different compilers and flags are important for code verification and validation

FLiT (Floating-point Litmus Tester)

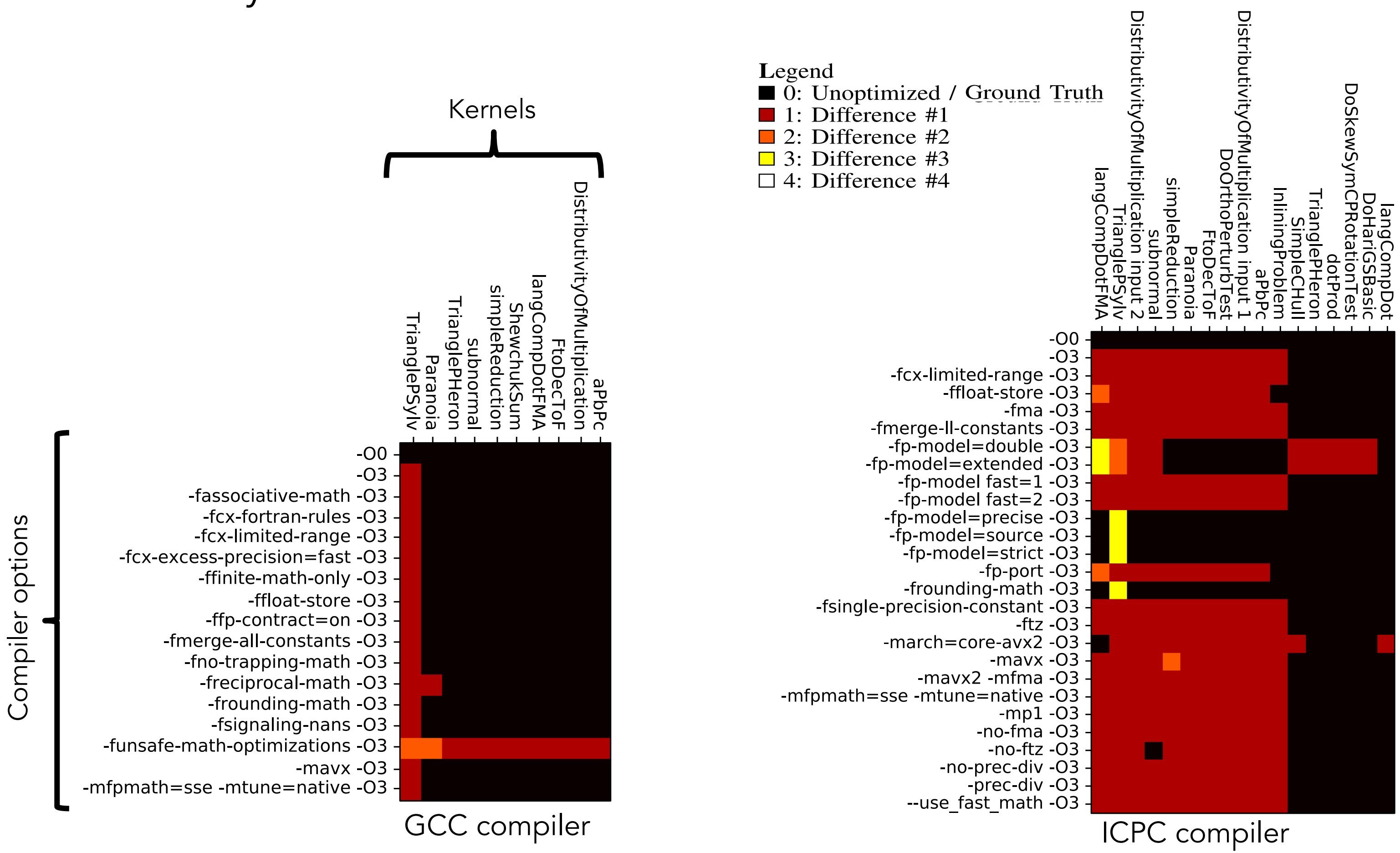
- FLiT is a reproducibility test framework
  – Test floating-point arithmetic variability in any user-given collection of programs
- FLiT tests the variability through hundreds of combinations
  – Different compilers, compiler flags, and also different hosts
- Results are stored in SQL database and used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



Visualization

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

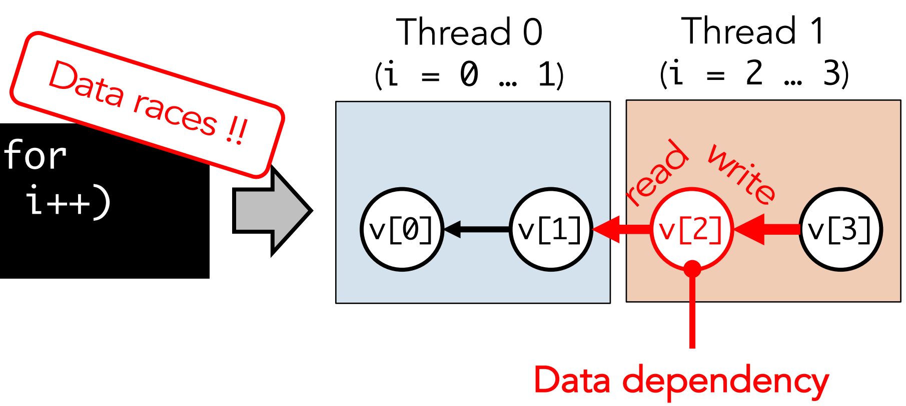


OpenMP easily creates non-deterministic bugs

- 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

Example

```
#pragma omp parallel for
for(int i = 0; i < 4; i++)
  v[i] = v[i+1] + 1;
```



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

Archer

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

OpenMP Source code → **Static Analysis** (OpenMP C/C++ Clang/LLVM Compiler) → **Dynamic Analysis** (Tsan and OpenMP RT) → **Report**

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

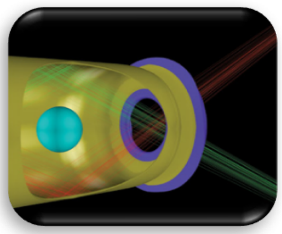
- Dynamic analysis
  - Detect data races based on LLVM/Clang ThreadSanitizer combined with OMPT-based annotation

Archer significantly reduce runtime overhead while providing high precision and accuracy

Archer case study: HYDRA

- Archer easily detected data races !

HYDRA (porting on Sequoia)



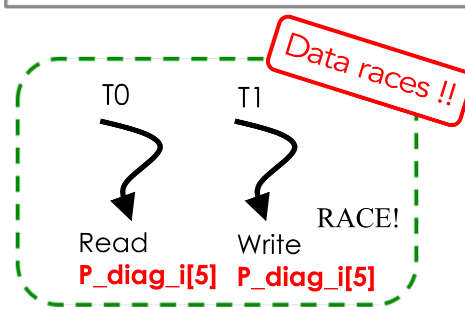
- MPI/OpenMP application non-deterministically crashed in an OpenMP region when compiling with optimization levels
- Manifested intermittently at or above 8K MPI processes
- The scientists spent months, and then ended up disabling OpenMP

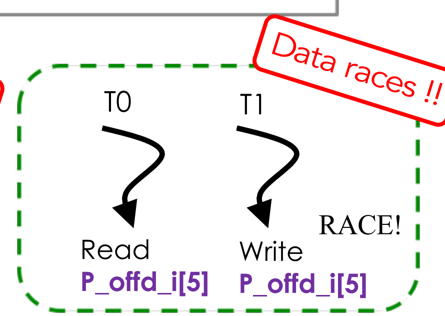
Archer

```
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];

1248:          P_diag_i[i] = cnt_diag;
1249:          P_offd_i[i] = cnt_offd;
…
1484:   } /* end parallel region */
1491:   return ieer;
1492: }
```

About 50 variables
423 lines





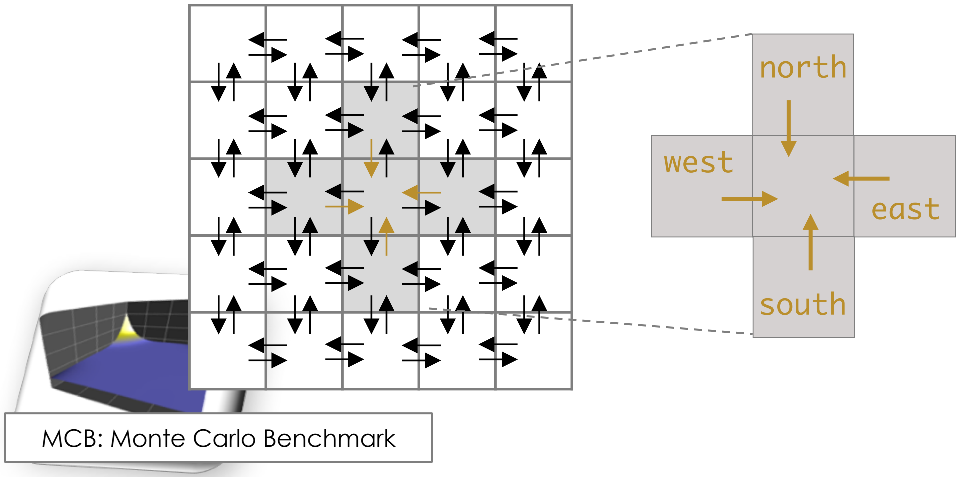
MPI can also introduce non-determinism

- It's typically due to communication with MPI_ANY_SOURCE
- In non-deterministic applications, each MPI rank doesn't know which other MPI rank will send message and when
- Example
  - If processes communication with neighbors, messages can arrive in any order from neighbors

MPI_ANY_SOURCE communication

```
MPI_Irecv(…, MPI_ANY_SOURCE, …);
while(1) {
  MPI_Test(flag);
  if (flag) {
    <computation>
    MPI_Irecv(…, MPI_ANY_SOURCE, …);
  }
}
```

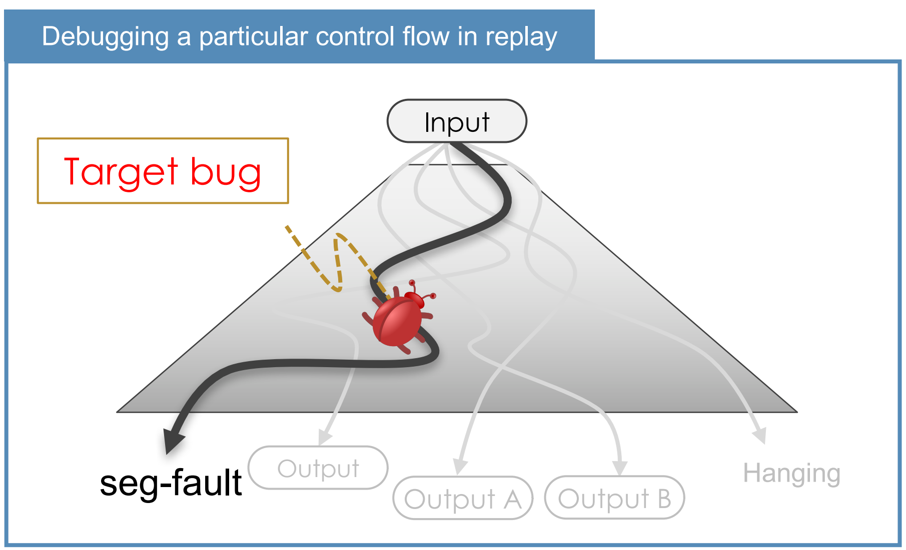
Example: Communications with neighbors



If a bug manifests through a particular message receive order,
It's hard to reproduce the bug, thereby, hard to debug it

ReMPI deterministically reproduces order of message receives

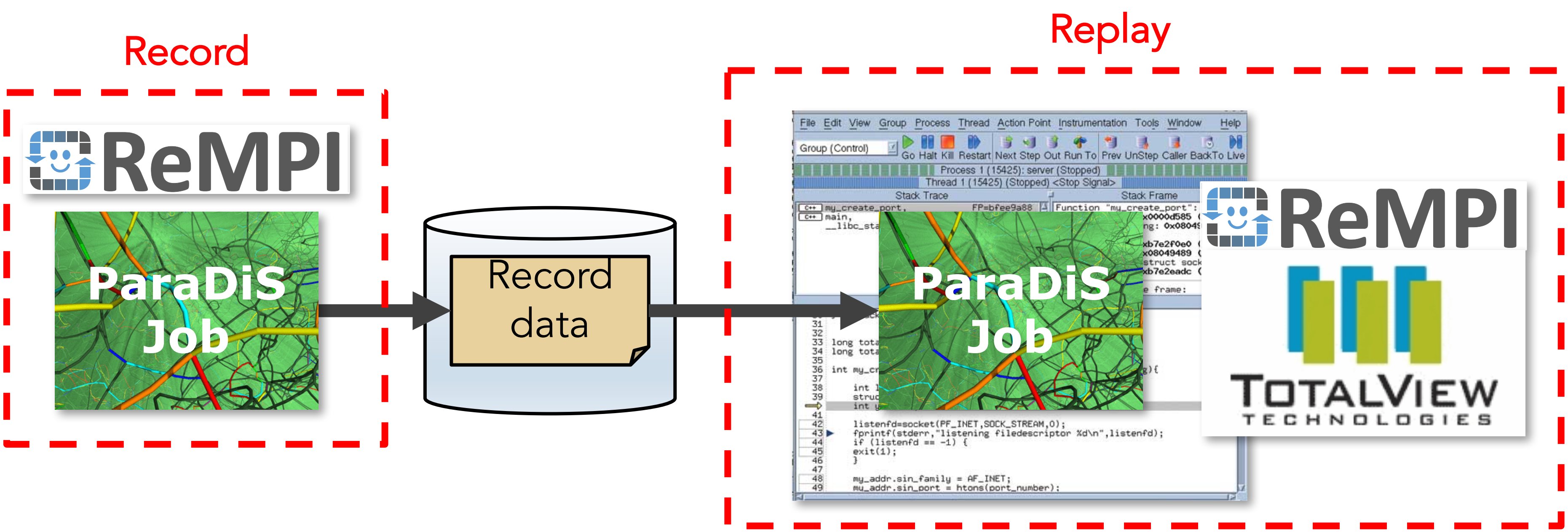
- ReMPI is an MPI record-and-replay tool
  - Record an order of MPI message receives
  - Replay the exactly same order of MPI message receives
- Even if a bug manifests in a particular order of message receives, ReMPI can consistently reproduce the target bug



ReMPI

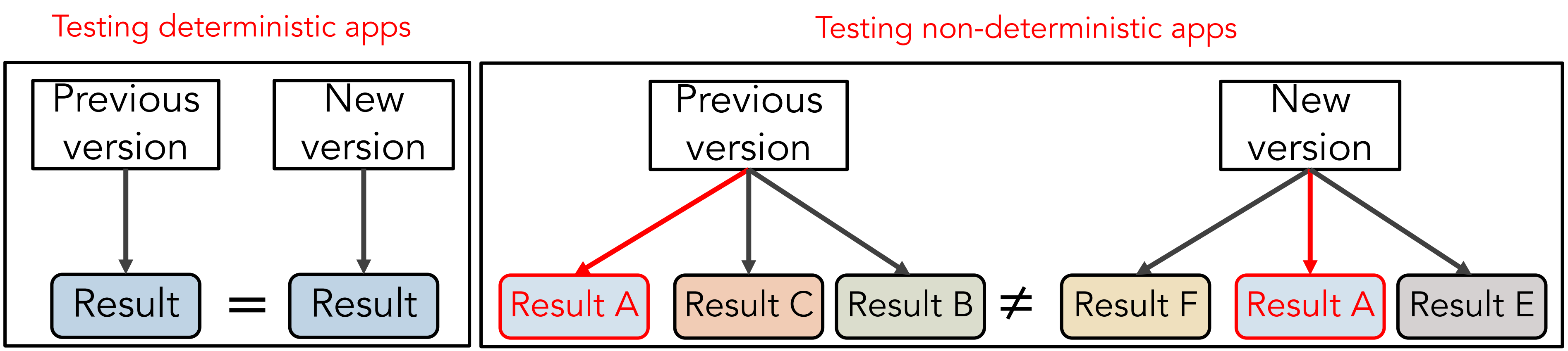
ReMPI case study: ParaDiS

- 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



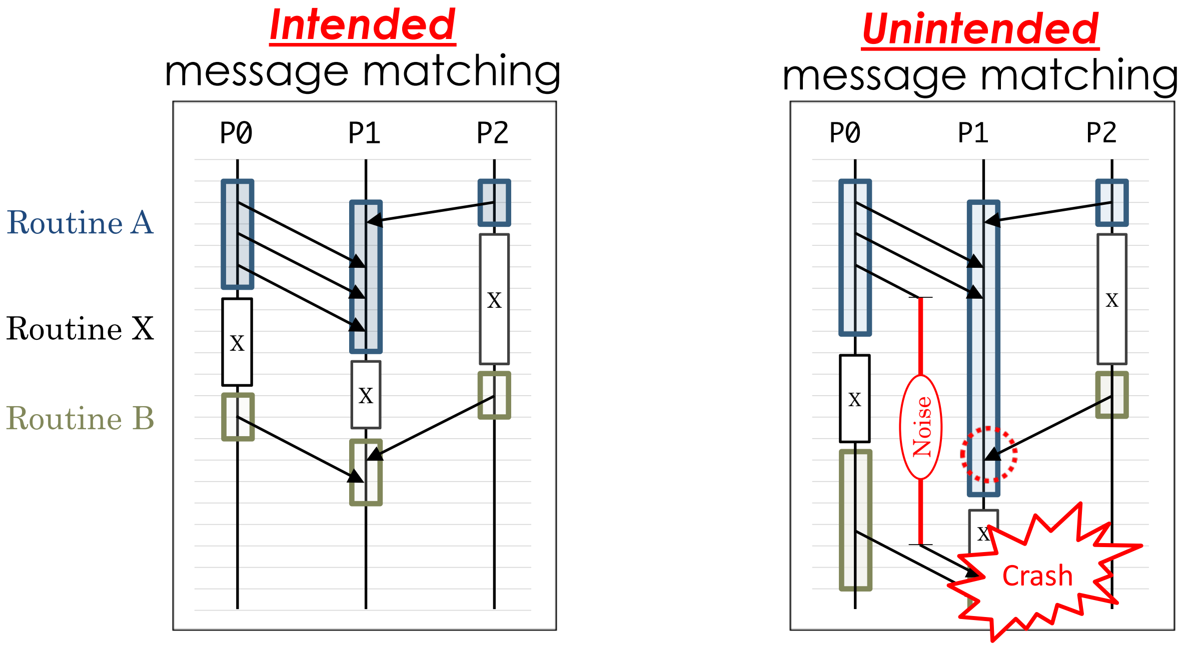
ReMPI is also useful for "Testing"

- "Testing" is also important for maintaining software quality
- MPI non-deterministic applications present significant challenges to testing
  - The non-determinism can produce different results from run to run by nature
- Using ReMPI, computational scientists can easily reproduce MPI behaviors, which facilitate testing



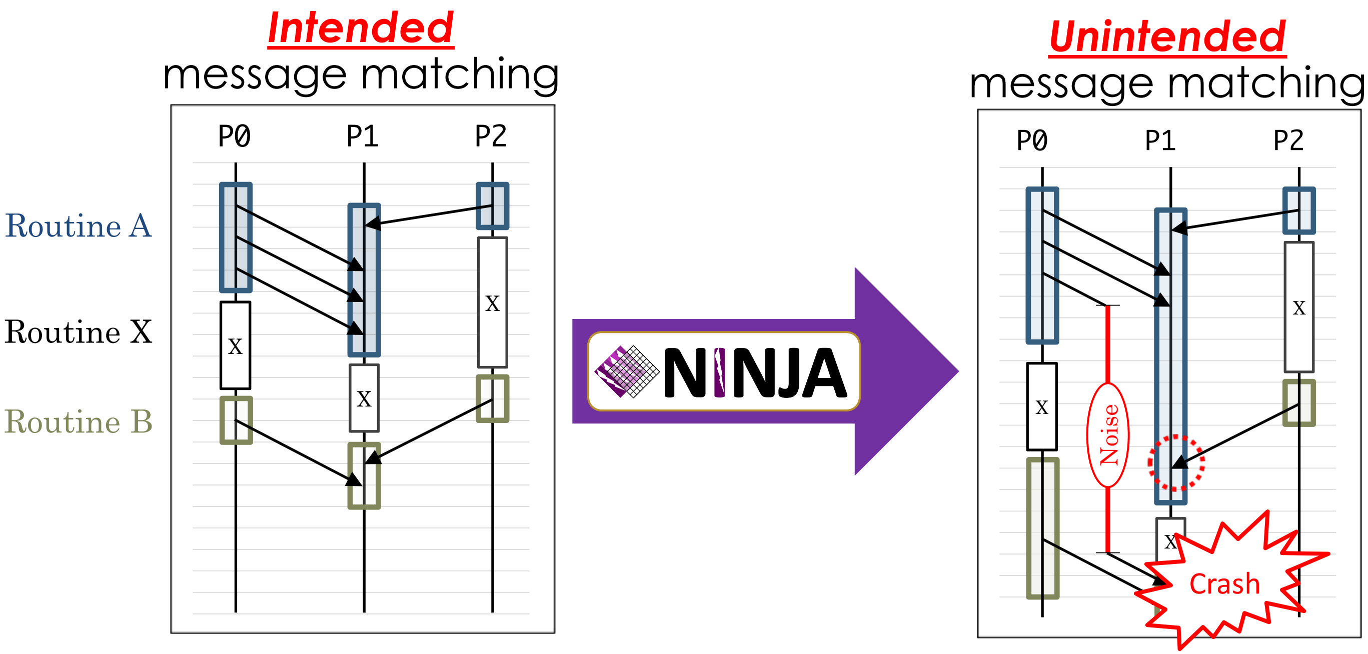
Unintended message races in MPI

- Many applications are written as a series of communication and computation routines (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

- NINJA exposes message race bugs by injecting noise

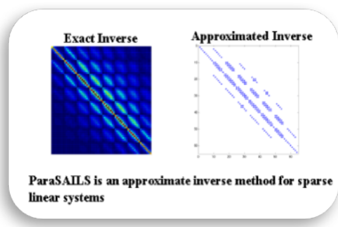


- NINJA detects suspicious communication routines
  - Communication routine using the same MPI_TAG without synchronization
- NINJA injects a delay to MPI messages in order to enforce message races
- NINJA can test if the application has unintended message races

NINJA cast study: Diablo/Hypre-2.10.1(in ParaSail module)

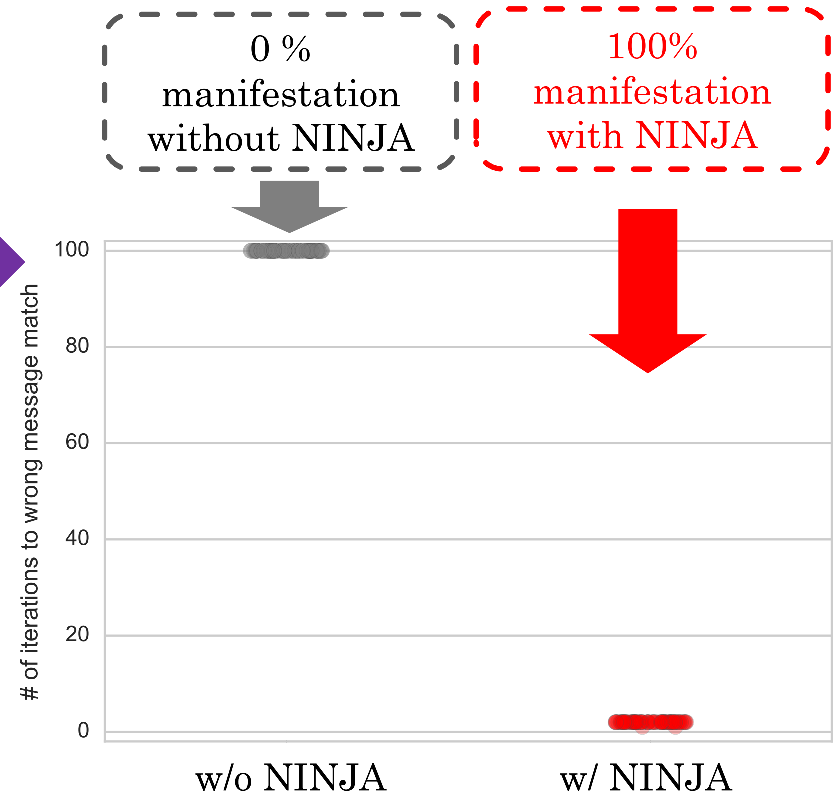
- Unintended message races in Hypre
- Prior to NINJA, the bug does not manifest itself in Hypre test code
- NINJA consistently exposed message races in the test code

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

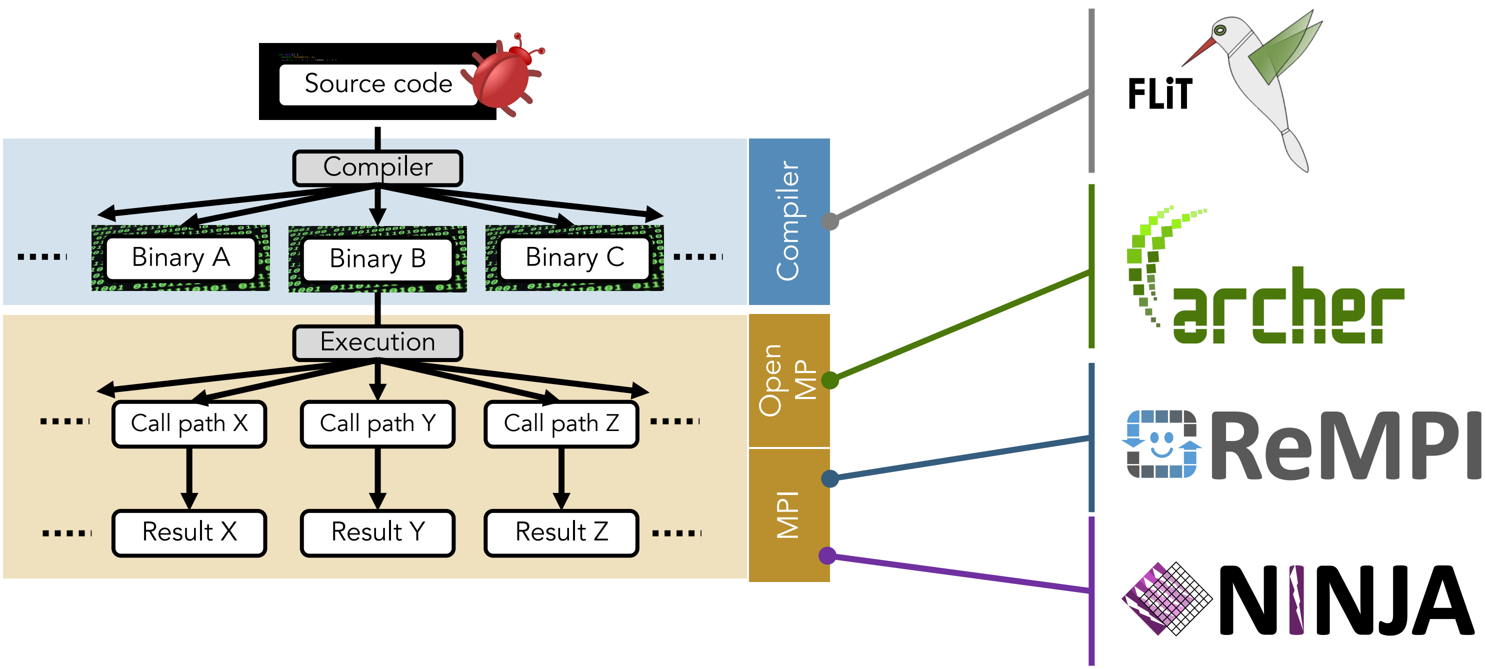
NINJA



Summary

- Debugging and testing large-scale HPC applications are becoming more challenging
- The PRUNERS toolset facilitates debugging and testing
- Exscale computing will be the culmination of non-deterministic execution for unprecedentedly high performance, and PRUNERS leads the way to solve its debugging and testing challenges.

PRUNERS



PRUNERS

https://pruners.github.io/

PRUNERS toolset

FLiT

archer

ReMPI

NINJA

Lawrence Livermore National Laboratory

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

The University of Utah

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

RWTH Aachen University

Joachim Protze

Lawrence Livermore National Laboratory