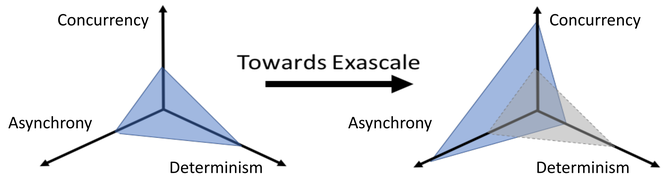
Towards Capturing Nondeterminism Motifs in HPC Applications

Dylan Chapp[1,2]
Advisors: Kento Sato[2], Dong H. Ahn[2], and Michela Taufer[1]
[1]University of Delaware, [2]Lawrence Livermore National Laboratory

Nondeterminism in HPC Applications

- Nondeterminism (ND) is increasingly embraced for scalability



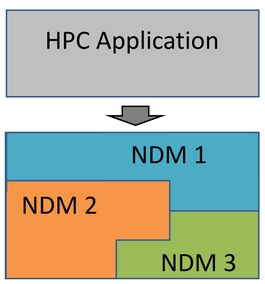
- Nondeterminism can lead to numerical and debugging side-effects for applications moving towards exascale

	Nondeterminism Sources	Side Effects
Recv side	• MPI_ANY_SOURCE • Nonblocking receives	• Numerical irreproducibility • Intermittent, scale-based bugs [1]
Send side	• Nondeterministic receive-dependent sends • Timing-dependent sends	• Incompatibility with tools and protocols that assume sender-determinism [2][3]

- Tools like PRUNERS[4] try to control some of the side-effects **BUT** an automatic identification of ND sources is missing
- Developers are forced to characterize ND by doing many runs with different *printf* debugging operations

Our Proposed Approach

- Identify and analyze representative nondeterminism snippets (small regions of source code)
- Build a classification system of ND patterns or *motifs* associated to these snippets to enable systematic control of nondeterminism's side-effects



***From* monolithic view of an applications**
- Limited ability to reason about ND
- Hard to debug and tools may not work

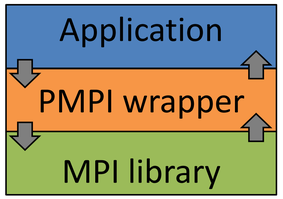
***To* a motif-based view of the application**
- Can reason about and debug effects of each nondeterministic motif (NDM)
- Know in advance which debugging tools will work

Our Target Applications

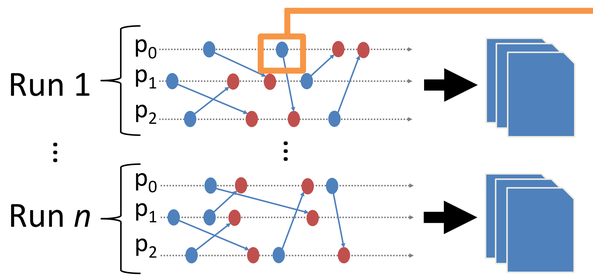
Application	Possible Recv–side Nondeterminism	Possible Send-side Nondeterminism
ParaDiS	Irecv + Waitany/all	Dynamic load-balancing based on wall-time
Monte Carlo Benchmark (MCB)	Irecv + Testsome	Unknown

Our Workflow for Capturing Nondeterminism Motifs

Step 1: Collect Traces from Multiple Runs



- Use MPI profiling API (PMPI)
  - Intercepts all send-initiation and receive-completion function calls
  - Logs each function call signature and its associated call stack

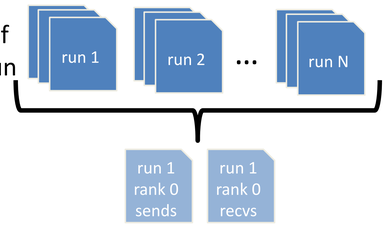


Example event record: isend signature and call stack

```
isend, idx=2, dst=2, cnt=59, tag=2010
0x2aaaab150ea2: (MPI_Isend+0xbf)
0x42cc05: (Buffered_MPI_Send::send(…)+0x85)
0x42c0ad: (Nonblocking_Sync::send_finished_internal(…)+0x5d)
0x41028b: (IMC::advancePhotonListDD(…)+0x8eb)
0x4105c1: (IMC::advance(…)+0x81)
0x2aaaad261aff: (GOMP_parallel+0x3f)
0x40574e: (main+0xe9e)
0x2aaaad6a7b35: (__libc_start_main+0xf5)
0x40656f: (_start+0x29)
```

Step 2: Detect Nondeterministic Events

- Collect sequences of events from each run
- Separate sequence into send-initiation events and receive-completion events



- Identify deterministic sequences and nondeterministic sequences
- Example: We identify run-to-run variations in sequences of MPI_Isend calls ([tag, dest, count])

Event	Run 1	Run 2	Run 3
0	[55, 2, 3360]	[55, 2, 3360]	[55, 2, 3360]
1	[55, 1, 3240]	[55, 1, 3240]	[55, 1, 3240]
2	[55, 1, 840]	[55, 1, 840]	[55, 1, 840]
3	[6502, 1, 1]	[6502, 1, 1]	[6502, 1, 1]
4	[6502, 2, 1]	[6502, 2, 1]	[6502, 2, 1]
5	[7294, 1, 1]	[7294, 1, 1]	[7294, 1, 1]
6	[7294, 2, 1]	[7294, 2, 1]	[7294, 2, 1]
7	[55, 2, 7200]	[55, 2, 7200]	[55, 2, 8400]
8	[55, 1, 7920]	[55, 1, 7920]	[55, 1, 7920]
9	[55, 2, 1320]	[55, 1, 1560]	[55, 1, 1680]
10	[55, 1, 1560]	[55, 2, 1800]	[55, 2, 600]
11	[55, 1, 240]	[55, 1, 240]	[6502, 1, 1]
12	[55, 2, 600]	[6502, 1, 1]	[6502, 2, 1]
13	[6502, 1, 1]	[6502, 2, 1]	[7294, 1, 1]
14	[6502, 2, 1]	[7294, 1, 1]	[7294, 2, 1]

- Only nondeterministic events' call stacks are passed on to next step

```
{ ND_event_1: callstack(ND_event_1),
  ND_event_2: callstack(ND_event_2),
  …
  ND_event_k: callstack(ND_event_k) }
```

Step 3: Associate Nondeterministic Events with Call-stacks

- Reduce ND events into a smaller set of call stacks
- Map each call stack to snippet (small regions of source code)
- Isolate source of ND by inspecting variables and data structures in snippets for local dataflow analysis

Example: MCB rank 0 sender nondeterminism

```
{ ND_event_1: callstack(ND_event_1),
  ND_event_2: callstack(ND_event_2),
  …
  ND_event_k: callstack(ND_event_k) }
```

Identify a single call stack resulting in all nondeterministic sends

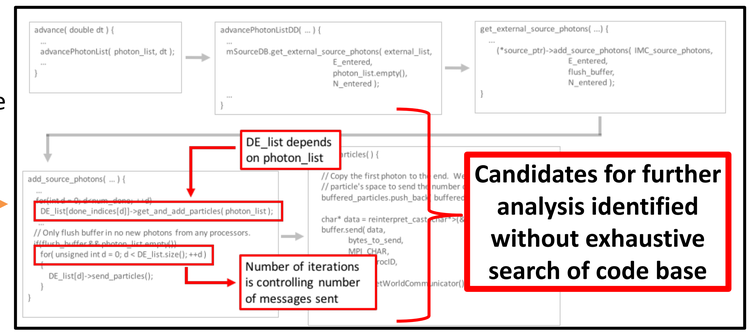
IMC_namespace::
IMC<…>::
advance

IMC_namespace::
IMC<…>::
advancePhotonListDD

IMC_namespace::
Source_data_base<…>::
get_external_source_photons

IMC_namespace::
Buffered_MPI_Send::
send

Map elements in call stack to source code snippets

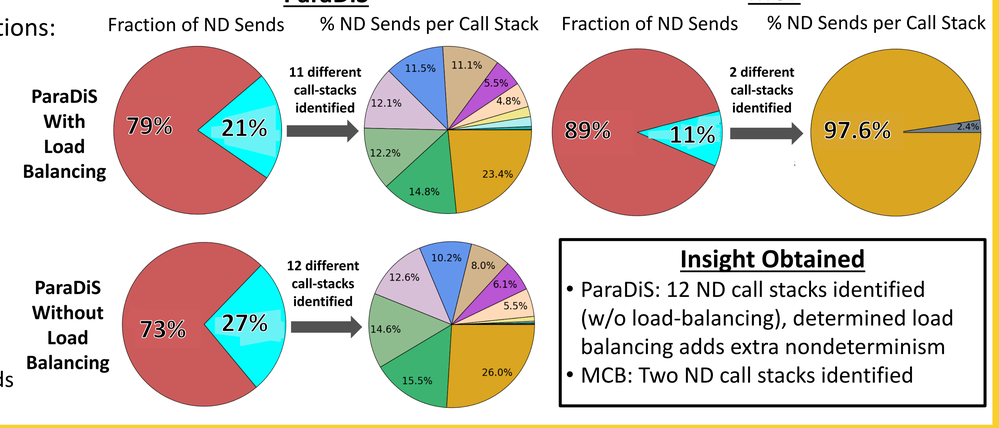


Results

- Workflow applied to two applications:
  - ParaDiS (with and without load-balancing)
  - Monte Carlo Benchmark (MCB)
- Metrics for quantifying nondeterminism:
  - Fraction of sends flagged as:

Deterministic or **Nondeterministic (ND)**

  - Number of call stacks resulting in nondeterministic sends
  - Fraction of nondeterministic sends associated with each call stack



Insight Obtained
- ParaDiS: 12 ND call stacks identified (w/o load-balancing), determined load balancing adds extra nondeterminism
- MCB: Two ND call stacks identified

Future Work

- Expand extraction of snippets for nondeterministic exascale codes
- Annotate exascale workflows with motifs to support reproducibility
- Use knowledge to mitigate runtime nondeterminism side-effects

References

1. K. Sato, D. Ahn, I. Laguna, G. Lee, and M. Schulz, "Clock delta compression for scalable order-replay of non-deterministic parallel applications", SC'15 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis
2. A. Guermouche, T. Ropars, M. Snir, F. Cappello, "HydEE: Failure containment without event logging for large scale send-deterministic MPI applications", IPDPS'12 Proceedings of the International Parallel and Distributed Processing Symposium
3. A. LeFray, T. Ropars, A. Schiper, "Replication for send-deterministic MPI HPC applications", FTXS'13 Proceedings of the 3rd Workshop on Fault-tolerance for HPC at Extreme Scale
4. D. H. Ahn, G. Lee, G. Gopalakrishnan, Z. Rakamaric, M. Schulz, I. Laguna, "Overcoming extreme scale reproducibility challenges through a unified, targeted, and multilevel toolset", SE-HPCCSE'13 Proceedings of the 1st International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract: DE-AC52-07NA27344 This work acknowledges NSF CCF 1513025 and the Exascale Computing Project 17-SC-20-SC

LLNL-POST-735705-DRAFT