Large-scale debugging with graphs

Nikoli Dryden (dryden2 <at> illinois <dot> edu)
University of Illinois at Urbana-Champaign
Advisor: Prof. Marc Snir (UIUC+ANL)

ESPT2015 Work-in-progress
How do we get a useful debugging experience at large scales?
What is a useful experience?

• Debugger works at the same scales as application

• Interactivity; rapid turn-around

• Intuitive input

• Command output and program state should be understandable and informative
What is a useful experience?

• Focus: Command output and program state should be understandable and informative

• ... And some scalability
Stack traces

- We know how to do this for stack traces
  - PGDB can merge similar stack traces
  - STAT has “3D trace/space/time” analysis
- Etc.
Outline

• Merging output
• Adding a notion of “time”: in general
• … And in specific
• Current status
Merging output

• Most ranks in application will be in (approximately) the same state

• Define equivalence classes for different types of output

• Merge output based on these and present overview
Merging output

print x

0
x=0

1
x=0

2
x=0

3
x=0xFF

[0-2]

x=0

3
x=0xFF
Merging output

- Tree-based reduction implements merging naturally
- Scalable and reduces data volume
Graph cuts

- Think of the graph of program execution (call tree)
- Each rank is somewhere in this when you get debugger output
- Regularly cutting the graph can provide context
- Merging now additionally considers which cut the output came from
Graph cuts
Collectives

• Some applications proceed in phases delineated by collective operations

• Simple choice for the cut point

• But not suitable for every application
PGDB

- Existing open-source parallel debugger for MPI
  https://github.com/ndryden/PGDB

- Basis for work

Current status

- Initial proof-of-concept implemented at small scale in PGDB
- Merging using MRNet filters
- But PGDB is currently text-based (not as pretty)
- Code available soon
Quick example

backtrace

...  
[0|1] #7 in PMPI_Reduce (...) at src/mpi/coll/reduce.c:1216
[0|1] #8 in advance (rank=0) at mpideadlock.c:14
[0|1] #9 in main (...) at mpideadlock.c:26
...
[1-2,4-15|2] #7 in PMPI_Reduce (...) at src/mpi/coll/reduce.c:1216
[1-2,4-15|2] #8 in advance (...) at mpideadlock.c:14
[1-2,4-15|2] #9 in main (...) at mpideadlock.c:26
...
[3|1] #2 in pthread_mutex_lock() from /lib64/libpthreads.so.0
[3|1] #3 in advance(...) at mpideadlock.c:9
[3|1] #4 in main(...) at mpideadlock.c:26
Future work

• Other notions for when to cut
• Handle MPI communicators better
• Further testing and scalability work
• Exploration: How well can we apply this to (lightweight) threads, etc.?
Thanks!

• Questions?