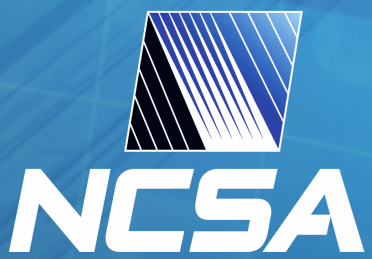
PGDB: A Debugger for MPI Applications

XSEDE '14

Nikoli Dryden

July 16, 2014



National Center for Supercomputing Applications

University of Illinois at Urbana-Champaign

Outline

1. Introduction
2. Overview of PGDB
3. Basic usage and example
4. Architecture of PGDB
5. Scalability
6. Recap and questions

Introduction

- Debugging is a necessary part of application development
- Good debuggers can help reduce development time

Introduction

- Debugging is a necessary part of application development
- Good debuggers can help reduce development time
- Parallel applications can be difficult to debug due to scale

Introduction

- Debugging is a necessary part of application development
- Good debuggers can help reduce development time
- Parallel applications can be difficult to debug due to scale
- When debugging at large scale, our debugger is a massively parallel application in its own right

Debuggers

- Start or attach to an application
- Stop an application, set breakpoints, watchpoints, etc.
- Examine/modify variables and memory
- Move through the execution
- Stack manipulation: backtraces, moving through frames, etc.
- Manipulate threads

Bugs at Large Scale

- Some errors only manifest themselves at large scale:

D. C. Arnold, et al. Stack trace analysis for large scale debugging. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1-10. IEEE, 2007.

Bugs at Large Scale

- Some errors only manifest themselves at large scale:
- Deadlocks
- Integer overflows, especially in counters
- Errors occur on different ranks each run
- Expect the unlikely

D. C. Arnold, et al. Stack trace analysis for large scale debugging. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1-10. IEEE, 2007.

Current Parallel Debuggers

- Parallel debuggers:
  - TotalView by Rogue Wave Software
  - DDT by Allinea
  - Eclipse Parallel Tools Platform
  - STAT by Paradyn

Current Parallel Debuggers

- Parallel debuggers:
  - TotalView by Rogue Wave Software
  - DDT by Allinea
  - Eclipse Parallel Tools Platform
  - STAT by Paradyn
- Kinda-parallel debuggers:
  - Launch GDB with `mpirun`
  - Manually use a debugger on one of the processes

Current Parallel Debuggers

- Parallel debuggers:
  - TotalView by Rogue Wave Software
  - DDT by Allinea
  - Eclipse Parallel Tools Platform
  - STAT by Paradyn
- Kinda-parallel debuggers:
  - Launch GDB with `mpirun`
  - Manually use a debugger on one of the processes
- "Debuggers": `printf`

PGDB

- Parallel GDB
- Debugger for MPI applications
- Builds upon GDB

PGDB

- Parallel GDB
- Debugger for MPI applications
- Builds upon GDB
- Free, open-source, scalable

PGDB

- Parallel GDB
- Debugger for MPI applications
- Builds upon GDB
- Free, open-source, scalable
- Basic idea: most processes that make up an application behave in a similar manner
- This lets us combine similar output for easier presentation and scalability

PGDB

- Parallel GDB
- Debugger for MPI applications
- Builds upon GDB
- Free, open-source, scalable
- Basic idea: most processes that make up an application behave in a similar manner
- This lets us combine similar output for easier presentation and scalability
- Stay similar to GDB for familiarity
- Use existing, robust tools for scalability (LaunchMON and MRNet)

Launching and Attaching

- Launch an application under PGDB's control:
  `pgdb [-l launcher] -a options`
- Attach to a running application:
  `pgdb -p pid`

Input/Output

- Like normal GDB, except MPI ranks
- Specify commands to run on a subset of ranks:

```
proc 0-16,48-64 print nd1i
[0-16,48-64] nd1i = 0
```

Input/Output

- Like normal GDB, except MPI ranks
- Specify commands to run on a subset of ranks:

```
proc 0-16,48-64 print nd1i
[0-16,48-64] nd1i = 0
```

- All the standard GDB commands (plus some non-standard ones)
- Filter output on remote nodes based on rules
- Pretty-print STL containers

Example

- `pgdb -a -n 1000 -f nodes ~/lulesh/lulesh2.0`

Example

- `pgdb -a -n 1000 -f nodes ~/lulesh/lulesh2.0`

```
PGDB deployed to 63 hosts and 1000 processors
[0-999] 0x00007f7e05124740 in __read_nocancel()
[0-999] Thread ID 1 stopped
```

Example

- `pgdb -a -n 1000 -f nodes ~/lulesh/lulesh2.0`

```
PGDB deployed to 63 hosts and 1000 processors
[0-999] 0x00007f7e05124740 in __read_nocancel()
[0-999] Thread ID 1 stopped
```

- `continue`

Example

- pgdb -a -n 1000 -f nodes ~/lulesh/lulesh2.0

```
PGDB deployed to 63 hosts and 1000 processors
[0-999] 0x00007f7e05124740 in __read_nocancel()
[0-999] Thread ID 1 stopped
```

- continue

```
[0-999] Thread ID 1 running
```

Example

- `pgdb -a -n 1000 -f nodes ~/lulesh/lulesh2.0`

```
PGDB deployed to 63 hosts and 1000 processors
[0-999] 0x00007f7e05124740 in __read_nocancel()
[0-999] Thread ID 1 stopped
```

- `continue`

```
[0-999] Thread ID 1 running
[1] Received signal SIGSEGV, Segmentation
fault
```

Example

- pgdb -a -n 1000 -f nodes ~/lulesh/lulesh2.0

```
PGDB deployed to 63 hosts and 1000 processors
[0-999] 0x00007f7e05124740 in __read_nocancel()
[0-999] Thread ID 1 stopped
```

- continue

```
[0-999] Thread ID 1 running
[1] Received signal SIGSEGV, Segmentation
fault
```

- proc 0,2-999 interrupt

Example

```
▶ pgdb -a -n 1000 -f nodes ~/lulesh/lulesh2.0
PGDB deployed to 63 hosts and 1000 processors
[0-999] 0x00007f7e05124740 in __read_nocancel()
[0-999] Thread ID 1 stopped
▶ continue
[0-999] Thread ID 1 running
[1] Received signal SIGSEGV, Segmentation
fault
▶ proc 0,2-999 interrupt
[0,2-999] Received signal 0, Signal 0
[0,2-999] Done.
```

Example

- `backtrace`

Example

- backtrace

```
...
[0,2-999] #3 0x00002b38b6fb34ac in PMPI_Waitall
at src/mpi/pt2pt/waitall.c:297
[0,2-999] #4 0x0000000000417563 in CommSend at
lulesh-comm.cc:843
[0,2-999] #5 0x00000000004071ee in
CalcQForElems at lulesh.cc:1997
...
```

Example

```
[1] #0 0x0000000000401abe in
CollectDomainNodesToElemNodes at lulesh.cc:261
[1] #1 0x000000000040fd58 in
CalcKinematicsForElems at lulesh.cc:1559
[1] #2 0x0000000000406d25 in
CalcKinematicsForElems at lulesh.cc:1538
[1] #3 0x0000000000406da4 in
CalcLagrangeElements at lulesh.cc:1612
...
```

Example

- `proc 1 info locals`

```
[1] nd1i = 0
...
[1] elemToNode = 0x0
...
```

Example

- `proc 1 info locals`

```
[1] nd1i = 0
...
[1] elemToNode = 0x0
...
```

- `proc 1 print elemToNode --frame 1`

```
[1] 0x0
```

PGDB's Architecture

- Challenges when building a massively parallel debugger:
  - Manage and analyze the large volume of data
  - Present results in a scalable manner
  - Use appropriate data structures from the start
  - Don't store unnecessary data

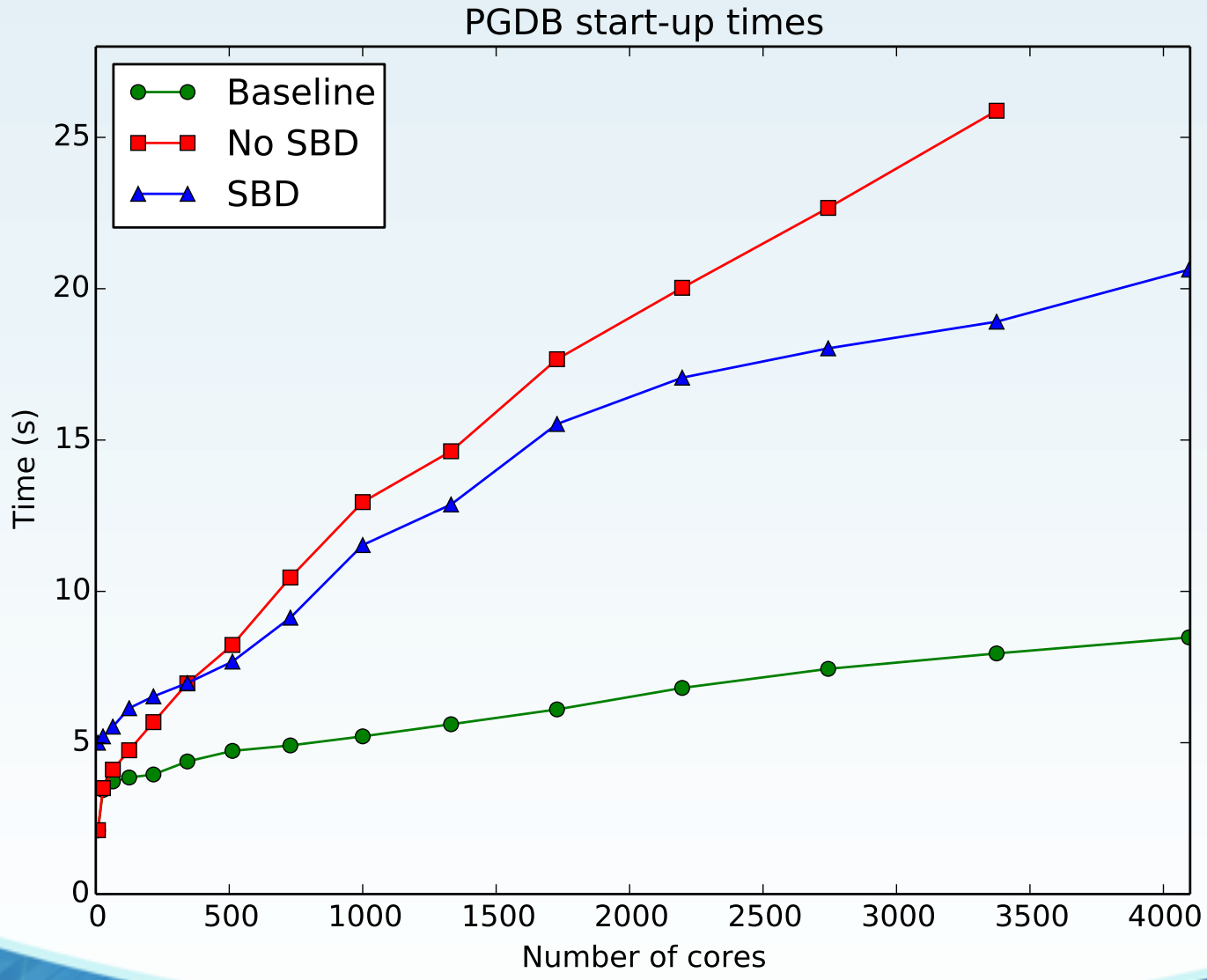
G. L. Lee, et al. Lessons learned at 208k: towards debugging millions of cores. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008.* IEEE, 2008.

PGDB's Architecture

- Challenges when building a massively parallel debugger:
  - Manage and analyze the large volume of data
  - Present results in a scalable manner
  - Use appropriate data structures from the start
  - Don't store unnecessary data
- Deployment outline:
  - Deploy lightweight tool daemons to remote nodes with LaunchMON
  - Bootstrap MRNet communication layer and deduplication filters
  - Optionally use scalable binary deployment at large scales
  - Manage GDB instances

G. L. Lee, et al. Lessons learned at 208k: towards debugging millions of cores. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008.* IEEE, 2008.

Scaling on Stampede



Future Work

- Further testing and improve scalability
- Test on additional systems (Cray, Blue Gene)
- Support Intel Xeon Phis, maybe GPUs
- Support other programming models: Charm++, GASNET, etc.
- Interface work (GUI)

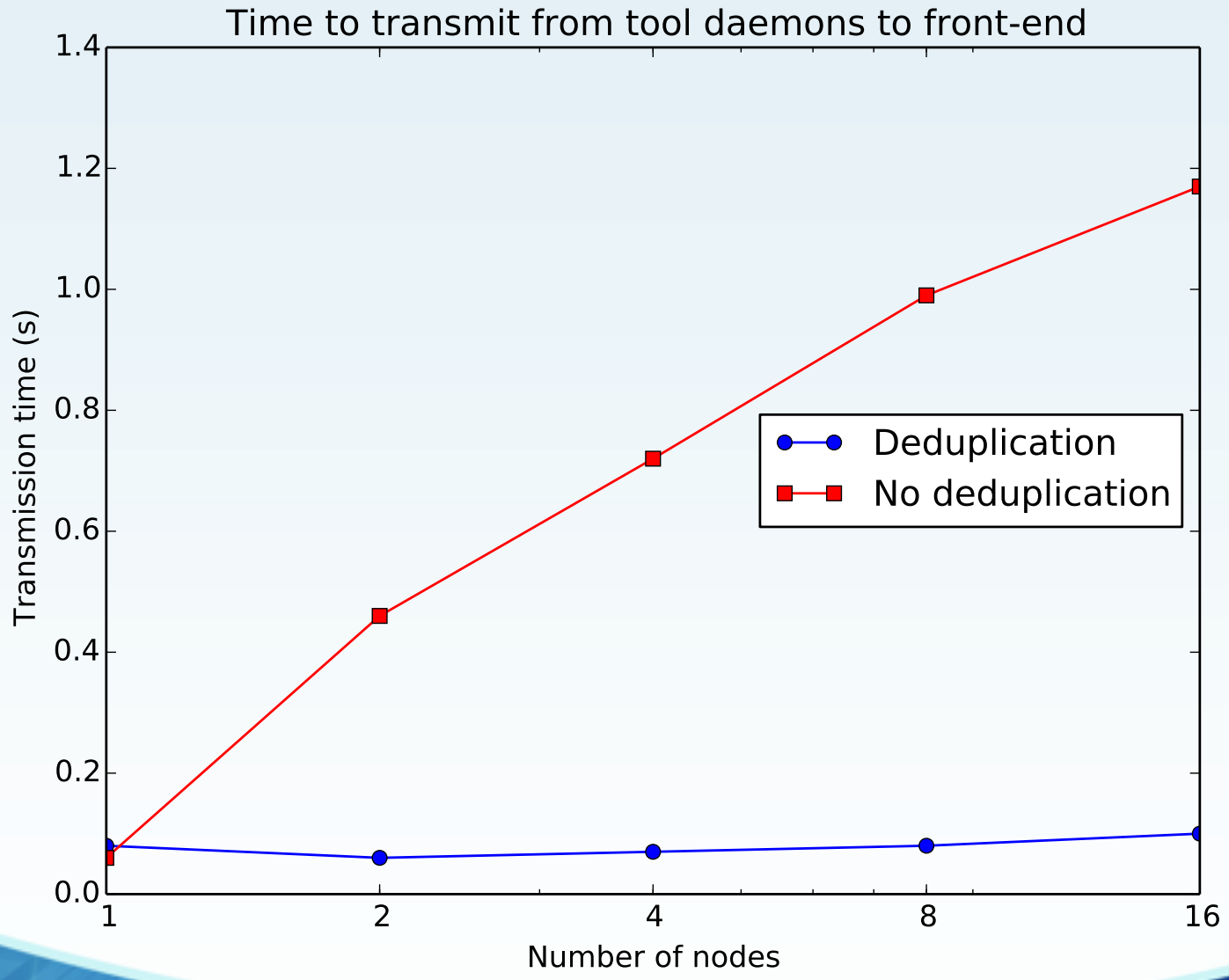
Recap

- Parallel debugging is important
- PGDB provides an alternative to existing solutions that is free, open-source, scalable
- Basic idea: most processes that make up an application behave similarly
- Build on existing approaches to create a robust and scalable tool

Questions?

- Acknowledgements:
  - This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575
  - Aaron Black and Mike Owen of LLNL
  - Dan LaPine and Alex Zahdeh of NCSA
  - Marc Snir of UIUC and ANL

Deduplication Filters



Deployment

- Built on LaunchMON
- Process acquisition with MPIR
- Launch to remote nodes
- Launch GDB instances
- Scatter topology information to set up main communication layer

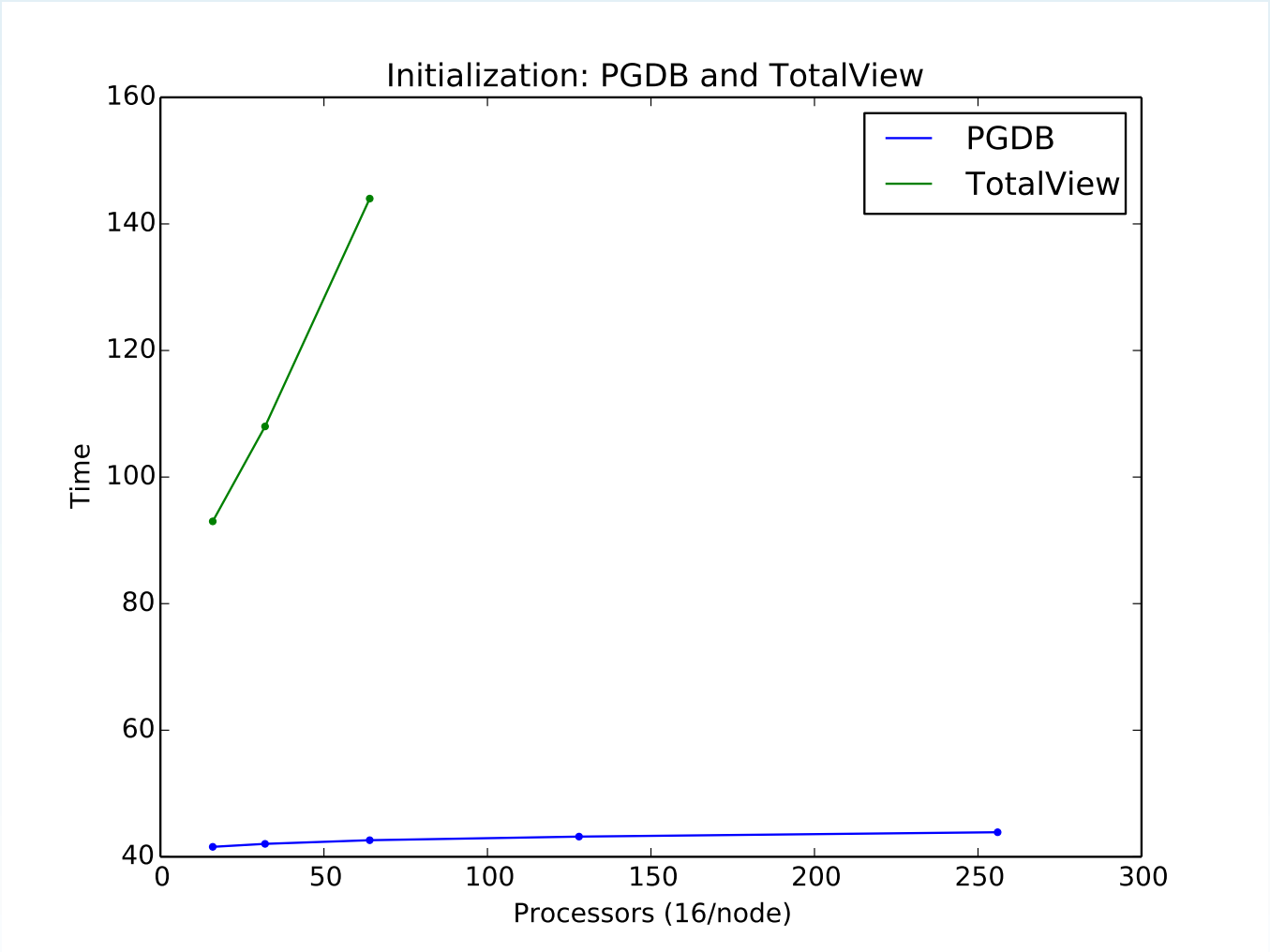
Communication

- Communication with MRNet
- Deployed as an n-ary tree, logarithmic height
- Deduplicate data with reduction filters

Scalable Binary Deployment

- Load files via our communication infrastructure instead of the parallel filesystem
- Helps with scaling to large numbers of processors

PGDB vs TotalView



URL

https://github.com/ndryden/PGDB