Linaro Connect
MADRID 2024 | MAY 12-17 2024

A GDB to support debugging High Performance Computing (HPC) Applications: Upstreaming

Richard Bunt, Principal Engineer, Linaro
Forge GDB Technical Lead

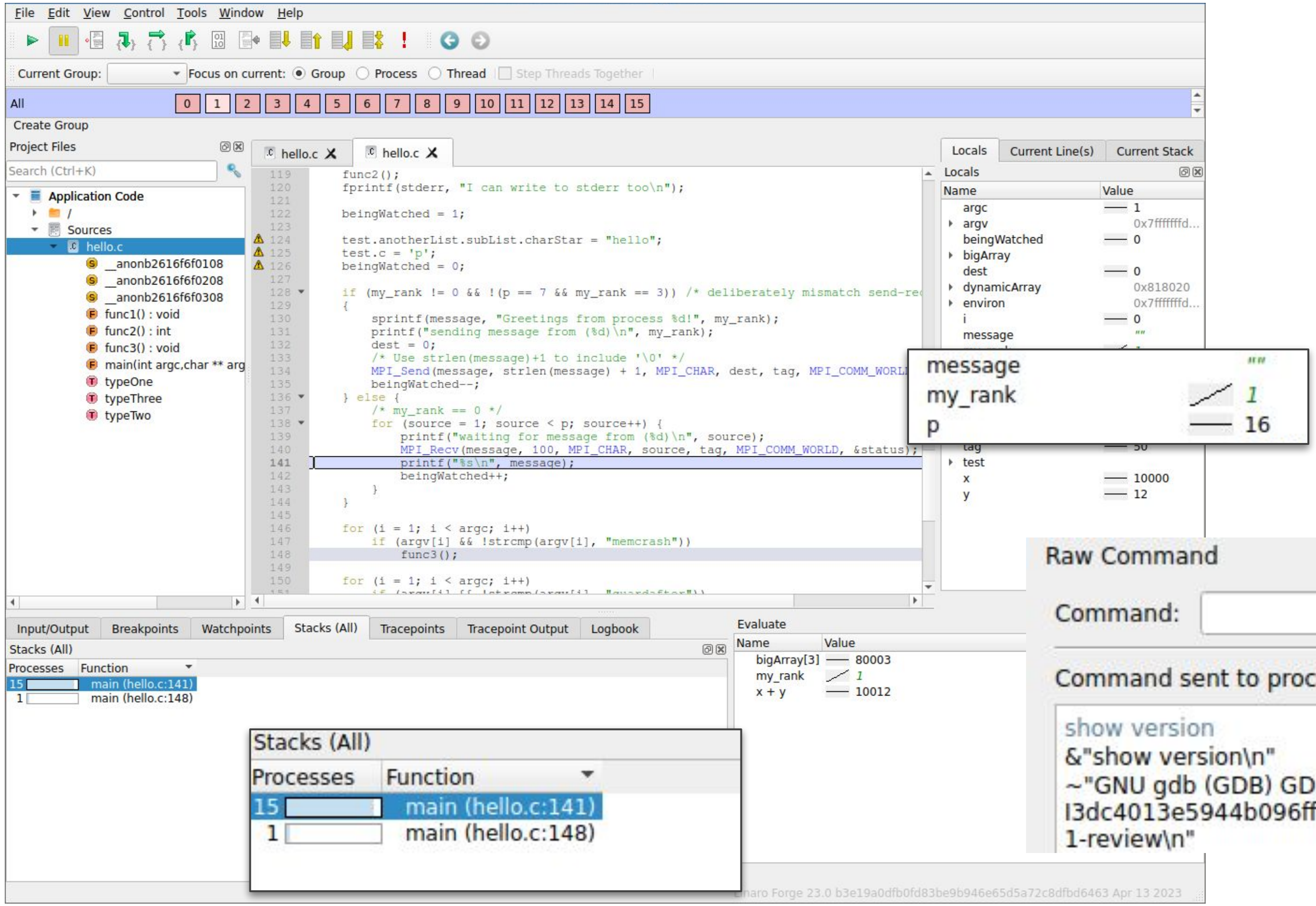
Agenda

- Inspired by a talk at Connect '23 around the benefits of upstreaming
- Similar story for Forge's High Performance Computing (HPC) extensions to GDB
- Background
  - High Performance Computing
  - Linaro DDT - A Graphical debugger for HPC
- GDB with HPC extensions
- Downstream Pain
- Relative Upstream Bliss

High Performance Computing (HPC)

- Parallel computing
- Examples:
  - Simulate galaxy creation, weather forecasting
  - Computational fluid dynamics or crash/impact simulations
- Commodity hardware (optimized) running Linux
- Languages {Fortran, C, C++, Python}
- Message Passing Interface (MPI)
  - Open MPI, MPICH, MVAPICH

Linaro DDT



```
119        func2();
120        fprintf(stderr, "I can write to stderr too\n");
121
122        beingWatched = 1;
123
124        test.anotherList.subList.charStar = "hello";
125        test.c = 'p';
126        beingWatched = 0;
127
128        if (my_rank != 0 && !(p == 7 && my_rank == 3)) /* deliberately mismatch send-re
129        {
130            sprintf(message, "Greetings from process %d!", my_rank);
131            printf("sending message from (%d)\n", my_rank);
132            dest = 0;
133            /* Use strlen(message)+1 to include '\0' */
134            MPI_Send(message, strlen(message) + 1, MPI_CHAR, dest, tag, MPI_COMM_WORL
135            beingWatched--;
136        } else {
137            /* my_rank == 0 */
138            for (source = 1; source < p; source++) {
139                printf("waiting for message from (%d)\n", source);
140                MPI_Recv(message, 100, MPI_CHAR, source, tag, MPI_COMM_WORLD, &status);
141                printf("%s\n", message);
142                beingWatched++;
143            }
144        }
145
146        for (i = 1; i < argc; i++)
147            if (argv[i] && !strcmp(argv[i], "memcrash"))
148                func3();
149
150        for (i = 1; i < argc; i++)
```

Locals

Name	Value
argc	1
argv	0x7fffffffd...
beingWatched	0
bigArray	
dest	0
dynamicArray	0x818020
environ	0x7fffffffd...
i	0
message	""
tag	50
test	
x	10000
y	12

message	""
my_rank	1
p	16

Stacks (All)

Processes	Function
15	main (hello.c:141)
1	main (hello.c:148)

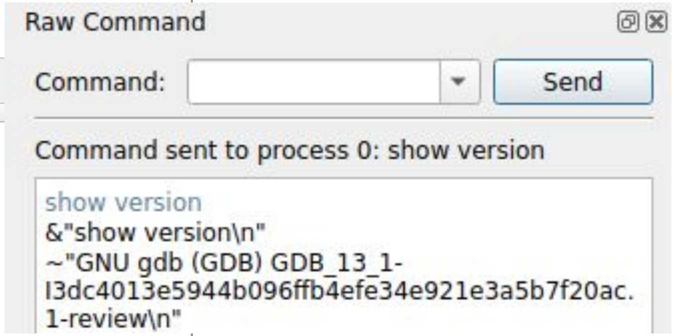
Evaluate

Name	Value
bigArray[3]	80003
my_rank	1
x + y	10012

Stacks (All)

Processes	Function
15	main (hello.c:141)
1	main (hello.c:148)

Linaro Forge 23.0 b3e19a0dfb0fd83be9b946e65d5a72c8dfbd6463 Apr 13 2023



Command sent to process 0: show version

```
show version
&"show version\n"
~"GNU gdb (GDB) GDB_13_1-l3dc4013e5944b096ffb4efe34e921e3a5b7f20ac.1-review\n"
```

GDB(s) for HPC

- Fortran (prevalent language in HPC)
- Non-GNU compiler support
- Stability
- Memory efficiency
- Third-party GPU GDBs

Example: Limited length printing

```
(gdb) print -elements 10 bigArray
$1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10…)

(gdb) set max-value-size 40
print -elements 10 bigArray
Value requires 40000 bytes, …
```

- HPC applications use large arrays to model physical effects multi-GB.
- GDB eagerly loads entire arrays
- Upstream status

Example: Fortran array slicing

```
!gdb/testsuite/gdb.fortran/array-slices.f90
$1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
print *,array(2:4)
$1 = (2, 3, 4)
print *,array(:3)
$1 = (1, 2, 3)
print *,array(5:3:-1)
$1 = (5, 4, 3)
print *,array4d(3:-2:-2,10:7:-2,:,-7:-10:-1)
…
```

- Array slicing syntax
- Inspect a small subset of an array
- 3 implementations
- Forge
  - 7 dimension limit
  - Limited slicing support
  - Memory efficient
- Upstream
  - Limited slicing support
- Fedora
  - Full slicing support
  - Memory inefficiency
- Upstream status

Example: Disable source file opening

- Reading source files from 10K GDBs from a shared file system
- GDB reads source files by default
- A similar issue affects init files
- set source open [on|off]
- Forge handles file access using its scalable tree.

Example: Max depth

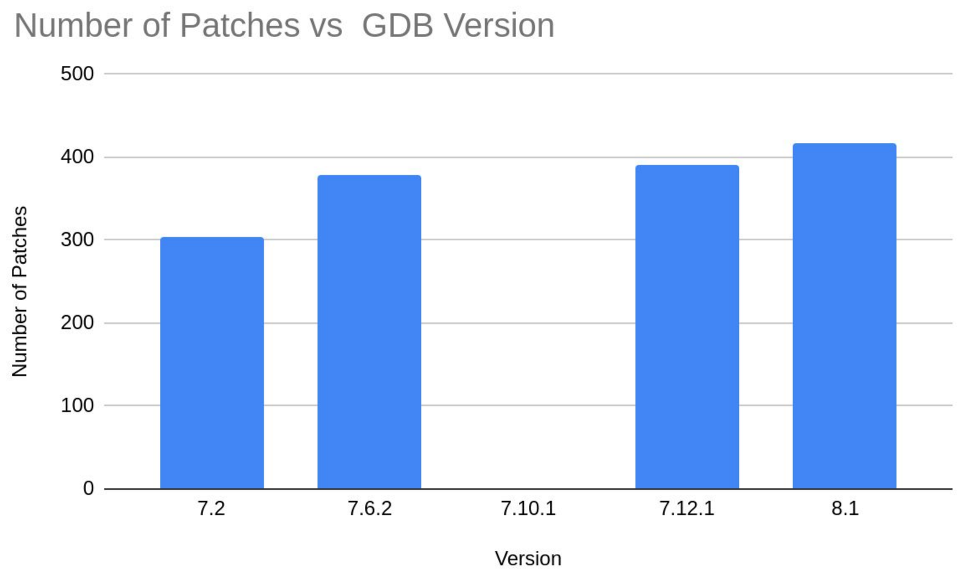
```
#define N 500
struct coordinate { int a; int b; int c; };
struct coordinates { int a[N]; int b[N]; int
c[N]; };

(gdb) print -max-depth 0 -elements 2 --
coordinates_i
$5 = {...}
(gdb) print -max-depth 1 -elements 2 --
coordinates_i
$6 = {a = {...}, b = {...}, c = {...}}
(gdb) print -max-depth 2 -elements 2 --
coordinates_i
$7 = {a = {0, 1...}, b = {0, 2...}, c = {0,
3...}}
```

- Control the amount of data
- SoA, AoS
- Combined with limited length
- Forge runs with a mix of 0 and 1
- Upstream status

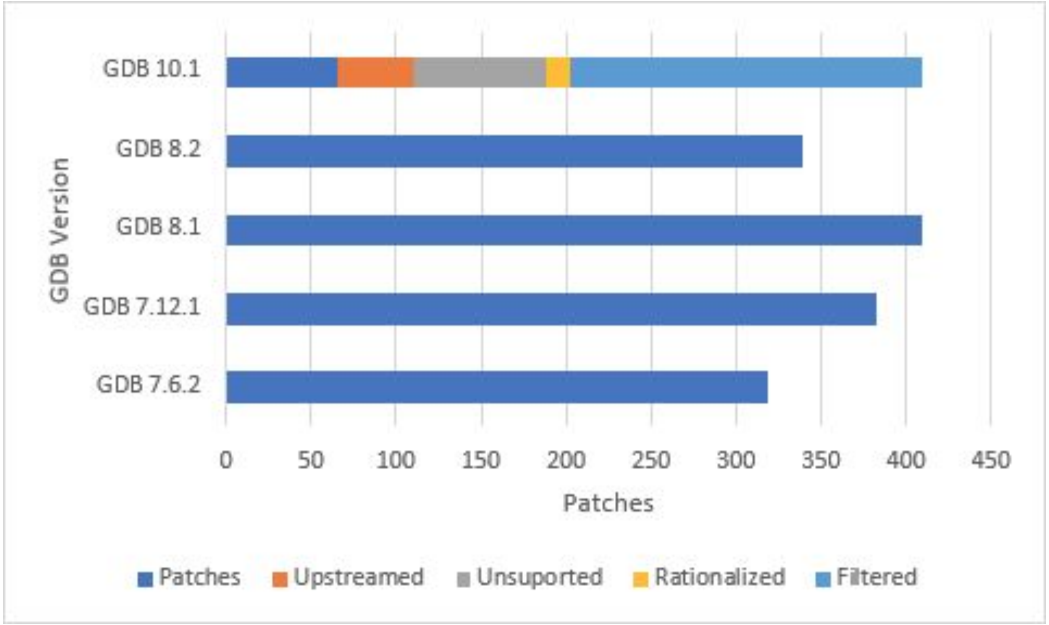
The Debt

- Patches vs. Lines of Code (LOC)
- GDB 7.6.2
  - Never again!
- GDB 7.10.1
  - Abandoned
- Number of patches increasing
- GPU GDBs
- Not sustainable



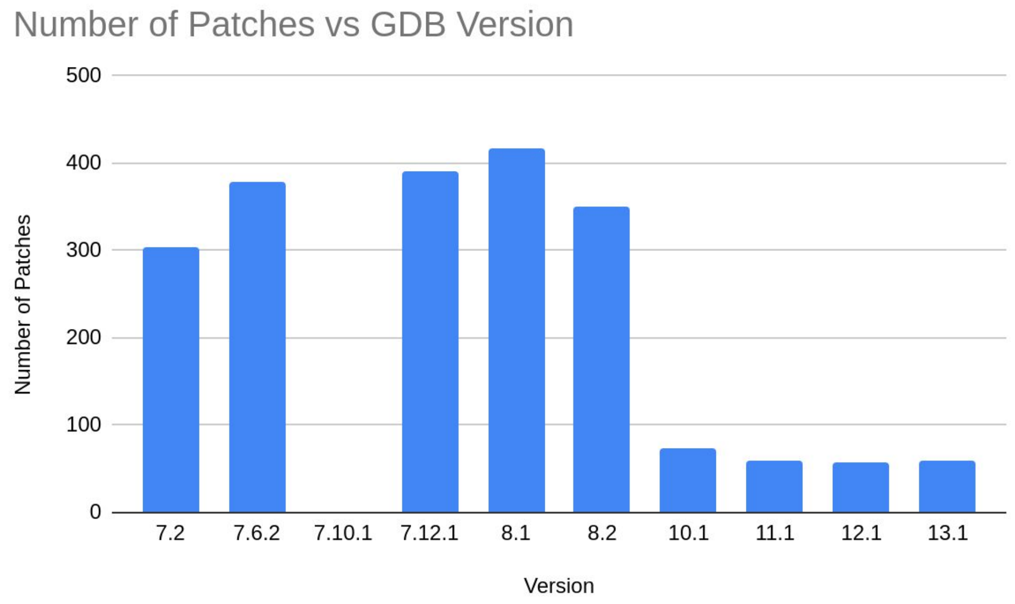
Patch Paydown

- Change of tack: Upstreaming
- Around GDB 9.1 that we had upstreamed enough
- Filtering



Now

- Further upstreaming
- Python plugins



Wins

- Everyone can benefit from the improvements
  - Previous examples
  - Support for isolating Python
- One Fortran array slicing implementation
- Reduced time to rebase from months to weeks
  - GDB 13
- Elided rebases altogether
  - System GPU GDBs
  - GDB 14
- Enabled customers earlier
  - Graviton 3 support case

Future

- Projecting GDB 15 to have even fewer patches
  - Upstreams
  - GDB plugins
- Continue upstreaming

Linaro Connect
MADRID 2024 | MAY 12-17 2024

Thank you

Any Questions?