Debugging at Scale

Comprehensive General PDC-KTH Course
Oct 12-13, 2023

Agenda

- Review of common signals
  - What the error message means
- Stack Trace Analysis Tool (STAT)
  - For when nothing appears to be happening…
- Abnormal Termination Processing (ATP)
  - For when things break unexpectedly… (Collecting back-trace information)
- GDB for HPC
  - Scaling the GDB debugger
- Valgrind for HPC
  - Valgrind-based debugging tool for parallel applications
- Sanitizers for HPC
  - Use several tools to check program correctness at run-time for parallel applications
- CRAY_ACC_DEBUG
 - For debugging GPU applications "for free"

The major types of bugs

- Crashing bugs
  - One or more processes in your application terminate
  - The most common kind
  - Generally (but not always) the easiest kind to solve
- Hangs
  - Deadlocks – everyone is stuck waiting for something that never happens
  - Livelocks – everyone is playing hot potato, calling different functions but not progressing
- Race conditions
  - One or more actors accessing the same data at the same time in a nondeterministic way
  - Shows up as changing results or sometimes crashes

Debugging in Production and Scale

- Even with the most rigorous testing, bugs may occur during development or production runs.
  - It can be very difficult to recreate a crash without additional information
  - Even worse, for production codes need to be efficient so usually have debugging disabled
- The failing application may have been using tens of or hundreds of thousands of processes
  - If a crash occurs one, many, or all of the processes might issue a signal.
  - We don't want the core files from every crashed process, they're slow to write and too big!
  - We don't want a backtrace from every process, they're difficult to comprehend and analyze.

Tools overview
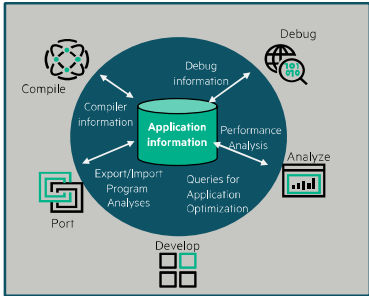
	Light weight No modification of the build chain, pre-/postprocessing. Get a first picture of a performance or problems during execution.	**In-depth** Requires modifications of the build system, recompile/relink, , pre-/postprocessing. Provides detailed information at user routine level.
Debugging Get your code up and running correctly.	**ATP** **STAT** **CRAY_ACC_DEBUG**	**gdb4hpc, (valgrind4hpc)** **Sanitizers4hpc** **Linaro (Arm) Forge Tools** **TotalView**
Profiling Locate performance bottlenecks.	**perftools-lite**	**perftools** **Apprentice2** **Reveal**

Review of common signals
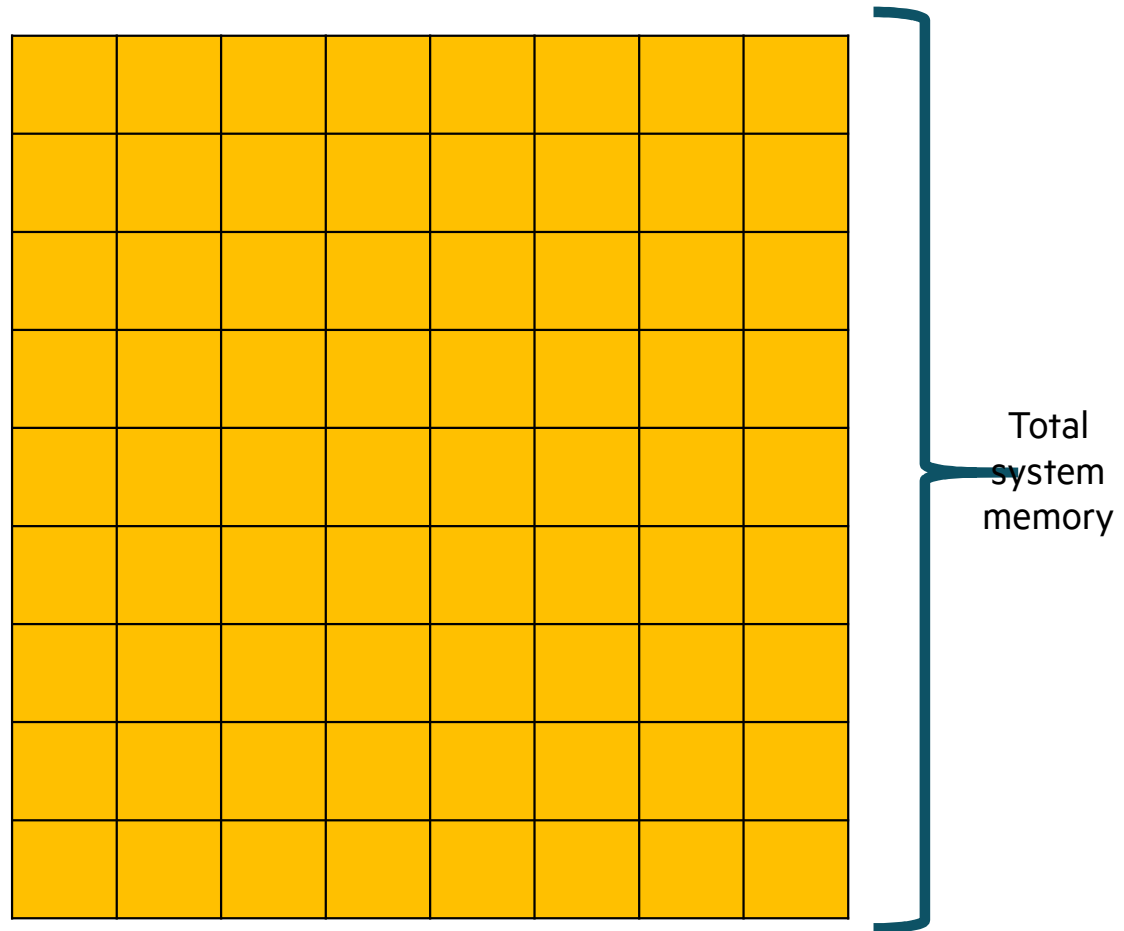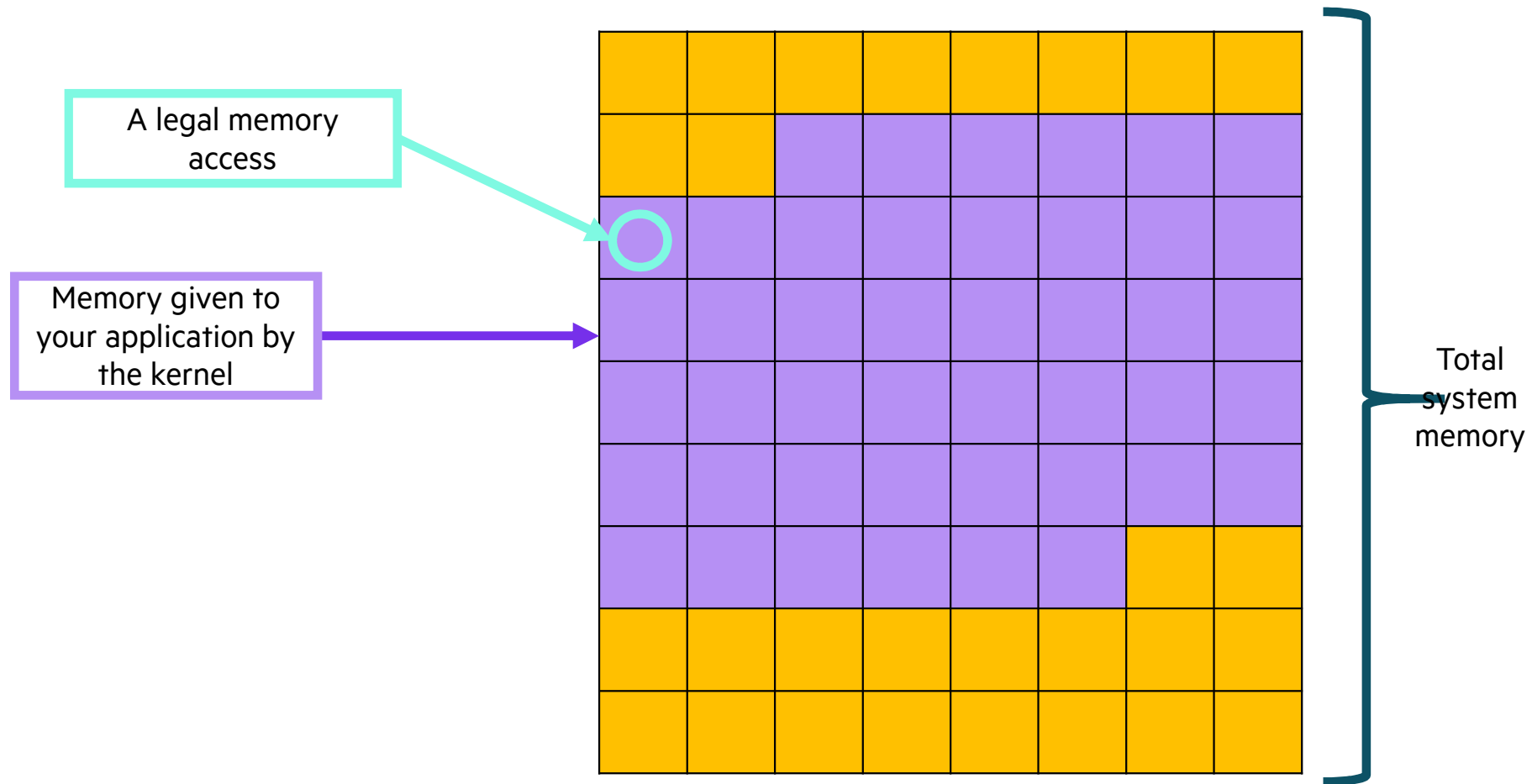
What the error message means

Common Signals from "crashes"

- "Words mean things"
- "man 7 signal" can act as a cheat sheet

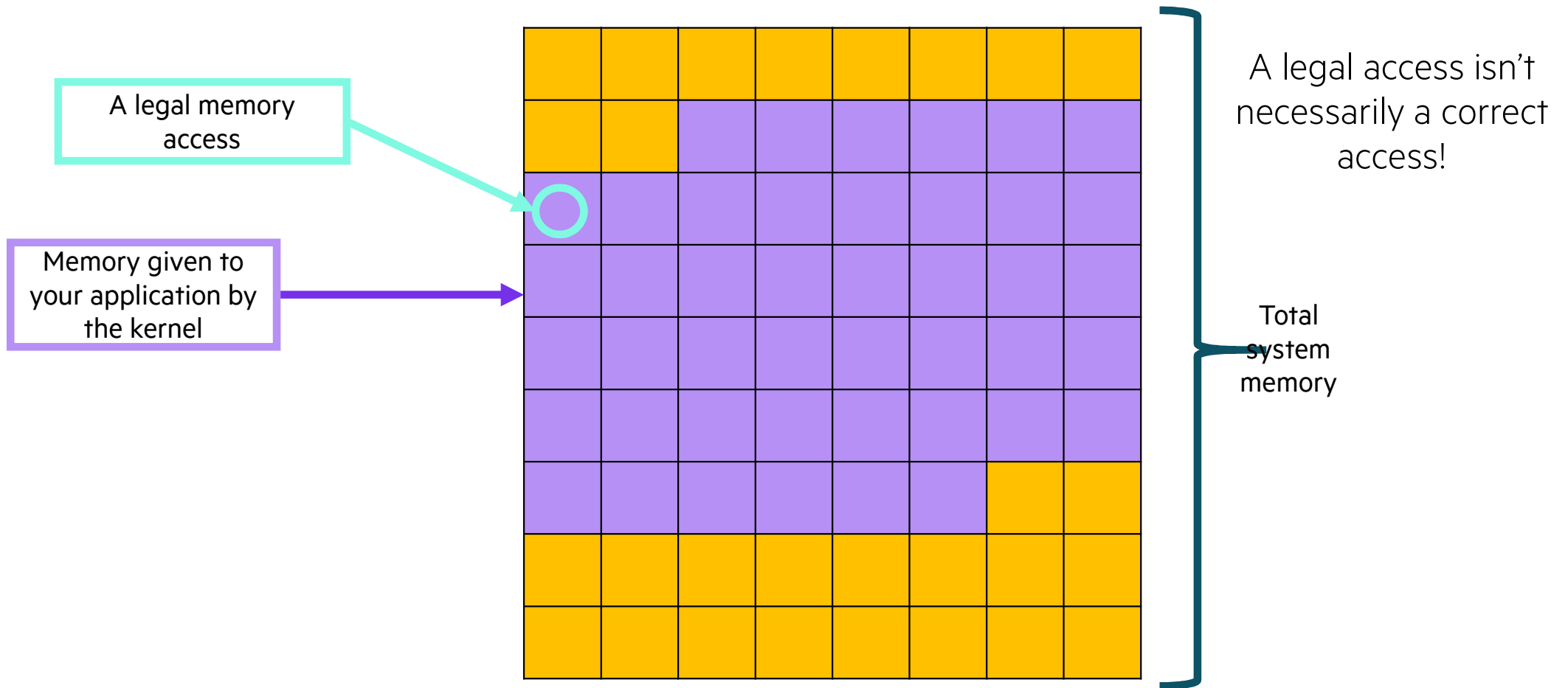| Signal Abbreviation (No. on x86/ARM)	Signal Name	What it means
SIGSEGV (11)	Segmentation Fault, AKA SegFault	You attempted to access memory that technically exists on the machine but is outside the virtual address space the kernel gave you
SIGBUS (7)	Bus error	You attempted to access memory that cannot possibly be accessed (most likely culprit nowadays: requirement for aligned memory not met)
SIGABRT (6)	Abort	Your application, or a library it uses, realized something was wrong and crashed intentionally
SIGFPE (8)	Floating Point Exception	You did some dangerous math and asked to be notified about it

SegFault/SIGBUS visualized

Total system memory

SegFault/SIGBUS visualized

A legal memory access

Memory given to your application by the kernel

Total system memory

SegFault/SIGBUS visualized

A legal memory access

Memory given to your application by the kernel

A legal access isn't necessarily a correct access!

Total system memory

SegFault/SIGBUS visualized

A legal memory access

Memory given to your application by the kernel

Total system memory

A segmentation fault

SegFault/SIGBUS visualized

A legal memory access

Memory given to your application by the kernel

A bus error

Total system memory

12

SegFault/SIGBUS visualized

A **non-aligned**, but legal memory access.

Memory given to your application by the kernel

A 16 bytes **aligned**, legal memory access

A bus error or segmentation fault

Total system memory

Offset: 0 1 2 3 4 5 6 7

SegFault/SIGBUS techniques

- Bounds checking
  - CCE Fortran `–hbounds`
  - Intel Fortran `–check bounds` or `--CB`
  - GNU Fortran `–fcheck=bounds` or `--fbounds-check`
  - C++ containers
- Address sanitizing (Use CPE sanitizers4hpc for parallel aggregation)
  - CCE C/C++/Fortran –fsanitize=address
  - LLVM –fsanitize=address
  - GNU C/C++ -fsanitize=address
- Valgrind （prefer ASAN) (valgrind4hpc for parallel aggregation)
- Core files/gdb/ATP

Things that abort

- Asserts
  - Consider assert(condition && "Descriptive string")
- Abort
  - C/C++
  - Fortran extension as a subroutine in most compilers
- Unhandled C++ exceptions

- NOT MPI_Abort unless you pass it SIGABORT
- NOT Fortran stop or exit
  - Consider using abort extension instead of stop on detected errors
    – Can core dump
    – Can reliably be caught by debuggers and tools

Trapping math and Clang

- Some compiler optimizations may generate intermediate arithmetic errors
- GNU compilers
  - DO NOT allow these optimizations at –O levels
  - ENABLE these with `-funsafe-math-optimizations` or `-fno-trapping-math`, but don't turn on trapping!
- Clang compilers
  - DO allow some of the optimizations at –O levels
  - DISABLE these with `-ffp-exception-behavior=maytrap`

Signals from an outside influence

- Some signals indicate the kernel or workload manager told your process to terminate
- "man 7 signal" can act as a cheat sheet

| Signal Abbreviation (Number)	Signal Name	What it means
SIGKILL (9)	Kill	Terminate immediately. Cannot be caught or handled. Often indicates out-of-memory.
SIGTERM (15)	Terminate	Please stop, but I'll give you time to clean up. Can be caught or ignored, but don't ignore for too long. Typically means allocation time has expired
SIGINT (2)	Interactive Interrupt	You C^c-ed on your terminal. Normally slurm will catch this and require a repeat to actually broadcast it
SIGBUS (7)	Bus error	Terminate immediately. Can indicate out-of-memory/disk-space when using mmap'ed files.

Stack Trace Analysis Tool (STAT)

For when nothing appears to be happening...

Description

- Gathers and merges stack traces from a running application's parallel processes.

- Creates call graph prefix tree
  - Compressed representation
  - Scalable visualization
  - Scalable analysis

- It is very useful when application seems to be stuck/hung

- Scales to many thousands of concurrent process.

- Available through the module `cray-stat`
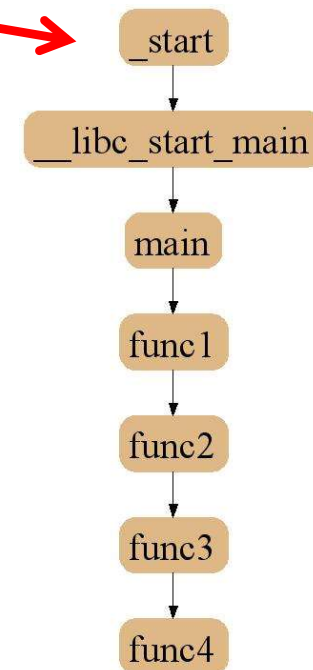
Stack Trace Merge Example
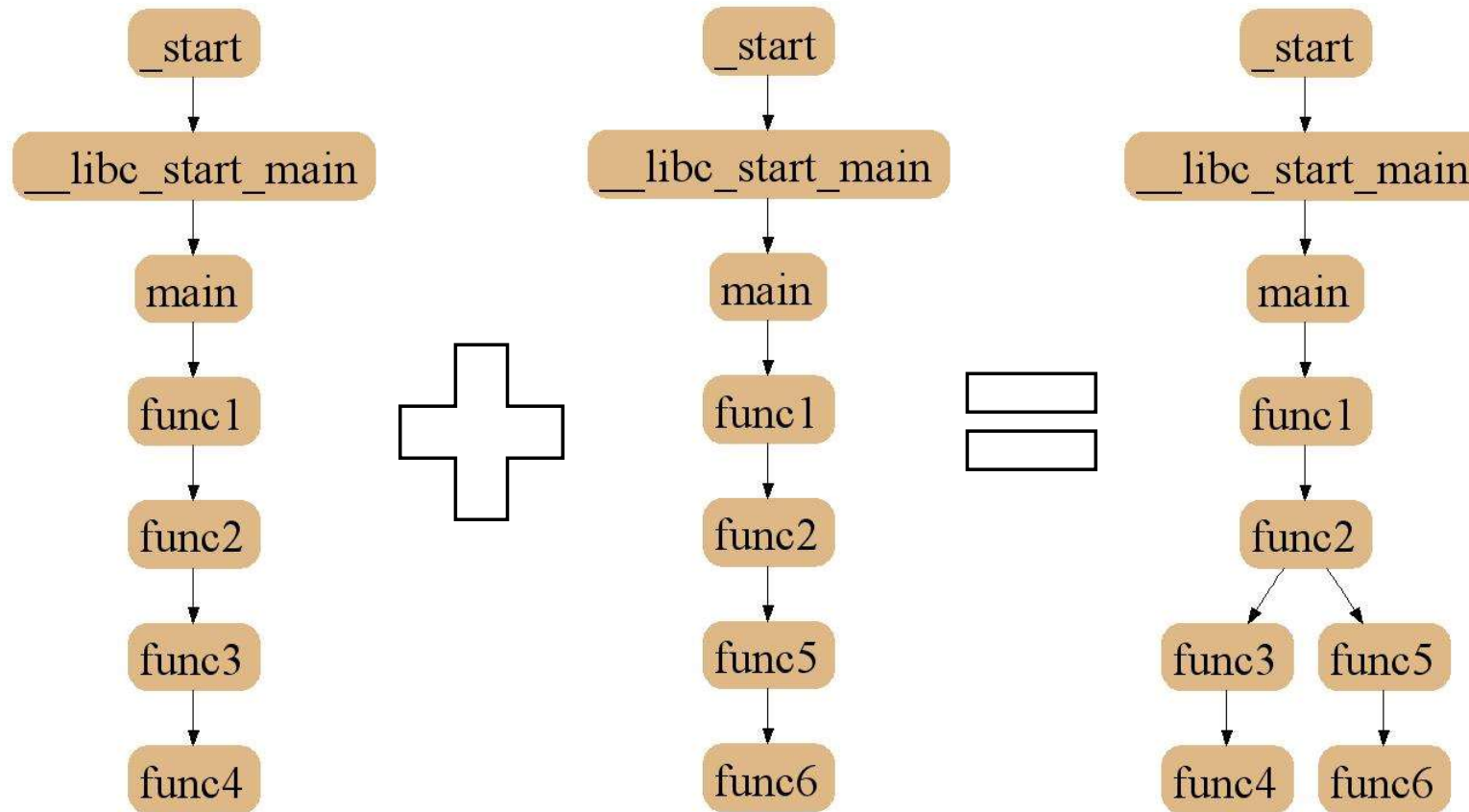
2D-Trace/Space Analysis
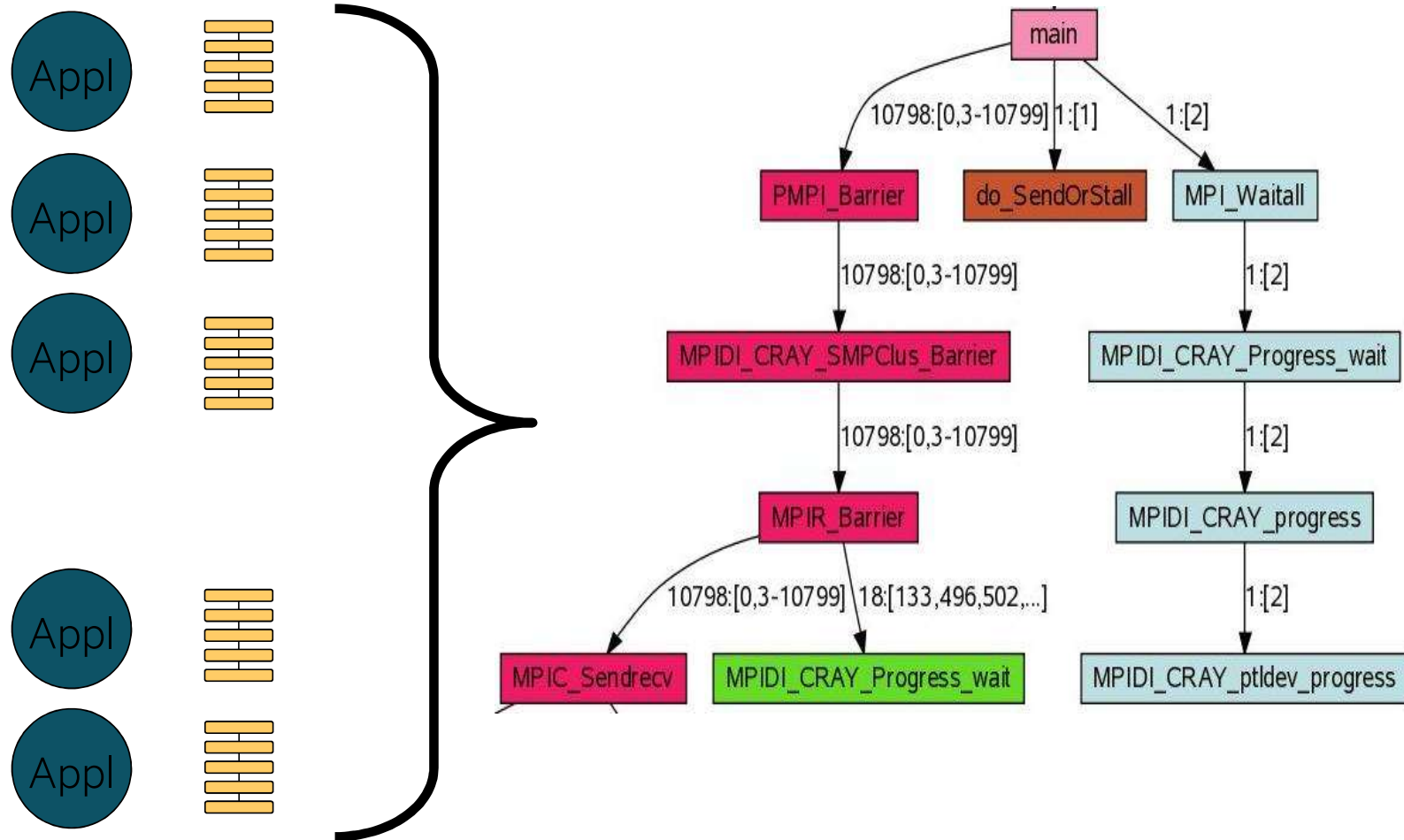
Using STAT from an Interactive Session

```
> module load cray-stat
> srun –n … ./<exe> &
```

- First get an interactive session (via salloc)
- Load the cray-stat module and run your application in the background

```
> stat-cl <pid_of_srun>
```

- Wait until application reaches the suspicious state
- Then launch the command line tool stat-cl with the process id of the **srun** as an argument and wait until it returns
- **Terminate the running application with** scancel **or** kill **the** srun

```
> stat-view stat_results/<exe>.0000/00.<exe>.0000.2D.dot
```

- **Now you can start the graphical interface** stat-view
- In order to use the graphical tool stat-gui to get the traces instead of the stat-cl please refer to **man** stat-gui
- More info: **man** stat-cl, **man** stat-view, and **man** stat-gui

Merged Stack
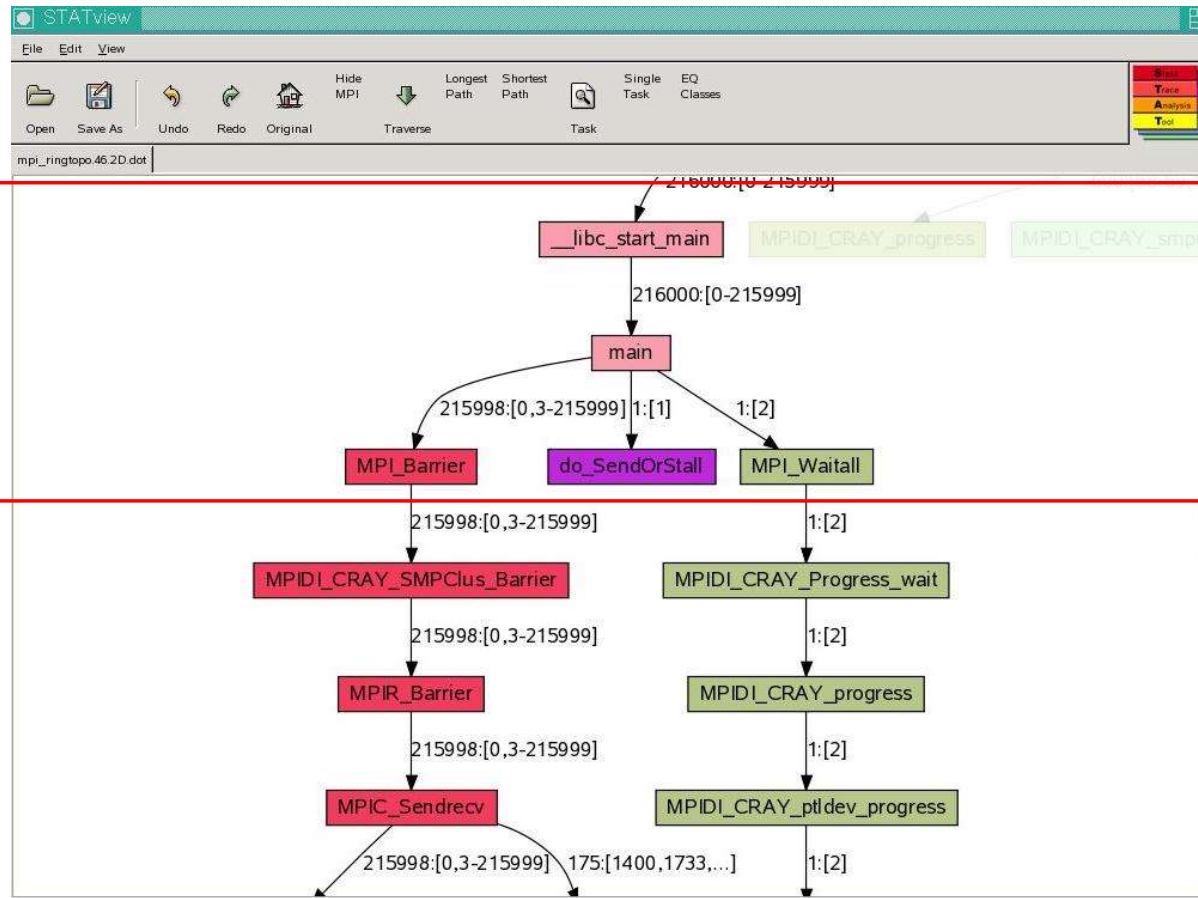
Using STAT in a Batch Job

```
module load cray-stat
stat-cl –s 30 –C srun –n … ./<exe>
```

- Prepend the `stat-cl` command to `srun` in your batch script
- The `–s <n>` option indicates that STAT should wait `<n>` seconds and then attempt to perform the snapshot. Therefore, this value should be sufficiently large that the application can be reliably assumed to be in the hung state

```
> stat-view stat_results/<exe>.0000/00.<exe>.0000.2D.dot
```

- **Now you can start the graphical interface** `stat-view`
- More info: `man stat-cl`, `man stat-view`, and `man stat-gui`

Abnormal Termination Processing (ATP)

For when things break unexpectedly...
(Collecting back-trace information)

Description

- Abnormal Termination Processing is a lightweight monitoring framework that detects crashes and provides more analysis instead of silently terminating.
  - Designed to be so light weight it can be used all the time with almost no impact on performance.
  - Almost completely transparent to the user
    - Requires `atp module` to be loaded during compilation (usually included by default)
    - Output controlled by the **ATP_ENABLED** environment variable (set by user, **ATP_ENABLED=1** for enabling it)
  - Tested at scale (tens of thousands of processors)

- ATP rationalizes parallel debug information into three easier to use forms:
  1. A single stack trace of the first failing process to stderr
  2. A visualization of every processes stack trace when it crashed
  3. A selection of representative core files for analysis

ATP Usage

```
export ATP_ENABLED=1
ulimit –c unlimited
module load atp
```

- Job scripts must include the changes above. Note that ATP respects ulimits on corefiles.
- After abnormal termination the application will not simply crash but proceed with the ATP analysis.
- Backtrace of first crashing process to stderr and the merged backtrace stored in dot files:
  - atpMergedBT.dot –Backtraces merged by function name only (smaller number of branches)
  - atpMergedBT_line.dot–Backtraces merged by function name and line number (more accurate but larger trace)

```
Application 867282 is crashing. ATP analysis proceeding...

Stack walkback for Rank 16 starting:
  [empty]@0xffffffffffffffff
  funcA@crash.c:8
Stack walkback for Rank 16 done
Process died with signal 11: 'Segmentation fault'
Forcing core dumps of ranks 16, 0
View application merged backtrace tree with: statview atpMergedBT.dot
You may need to: module load stat

_pmiu_daemon(SIGCHLD): [NID 00752] [c3-0c2s12n0] [Tue Feb 12 19:08:18 2013]
 PE RANK 0 exit signal Segmentation fault
[NID 00752] 2013-02-12 19:08:18 Apid 867282: initiated application terminat
ion
_pmiu_daemon(SIGCHLD): [NID 00753] [c3-0c2s12n1] [Tue Feb 12 19:08:18 2013]
 PE RANK 16 exit signal Segmentation fault
Application 867282 exit codes: 139
Application 867282 resources: utime ~2s, stime ~2s
slurm-10340.out lines 1-16/16 (END)
```
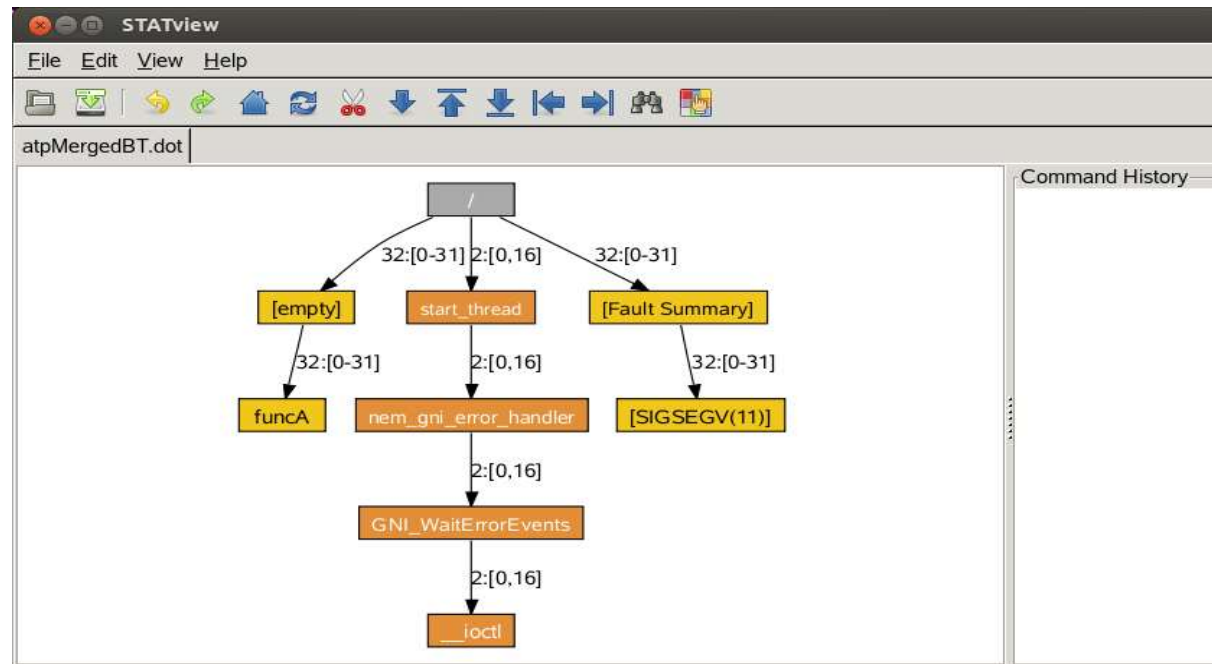
Trace back of crashing process

Core files are being generated.

Viewing the Results After the Crash

```
> module load cray-stat
> stat-view atpMergedBT.dot
```

- The merged backtrace is inspected via STAT.

- The core files can be inspected with **gdb** or `Linaro Forge`
- Man page apt for more info

# GDB for HPC

Scaling the GDB Debugger

Description

- The gdb debugger is now almost 40 years old
- It can load/start a process, attach to a running process or analyse a core dump
- It provides a command-line interface
- Requires compilation with debugging flags

Common commands (with shorter aliases):

- run (**r**): start or restart a program
- break (**b**): set a breakpoint location
- print (**p**): print the value of a variable or expression
- continue (**c**): continue running after a stop
- next (**n**): execute the next line of source code
- step (**s**): execute the next line, stepping into a function call
- backtrace (**bt**): show the current call stack
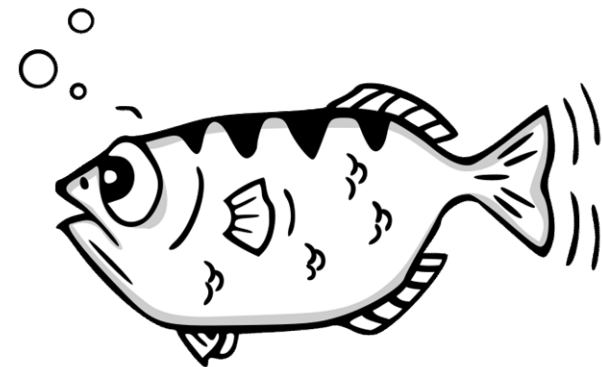
HTTPS://WWW.SOURCEWARE.ORG/GDB/IMAGES/ARCHER.SVG

GDB capabilities

- Watch points

- Signal and exception handling

- Conditional stops

- Executing commands at every stop

- Complex expressions (full interpreter)

- Making calls to program functions

- Interacting with threads

- Interpreting core files

- Custom pretty printing (via Python extensions)

- Examining raw data

- Built in shell with user variables and control structures
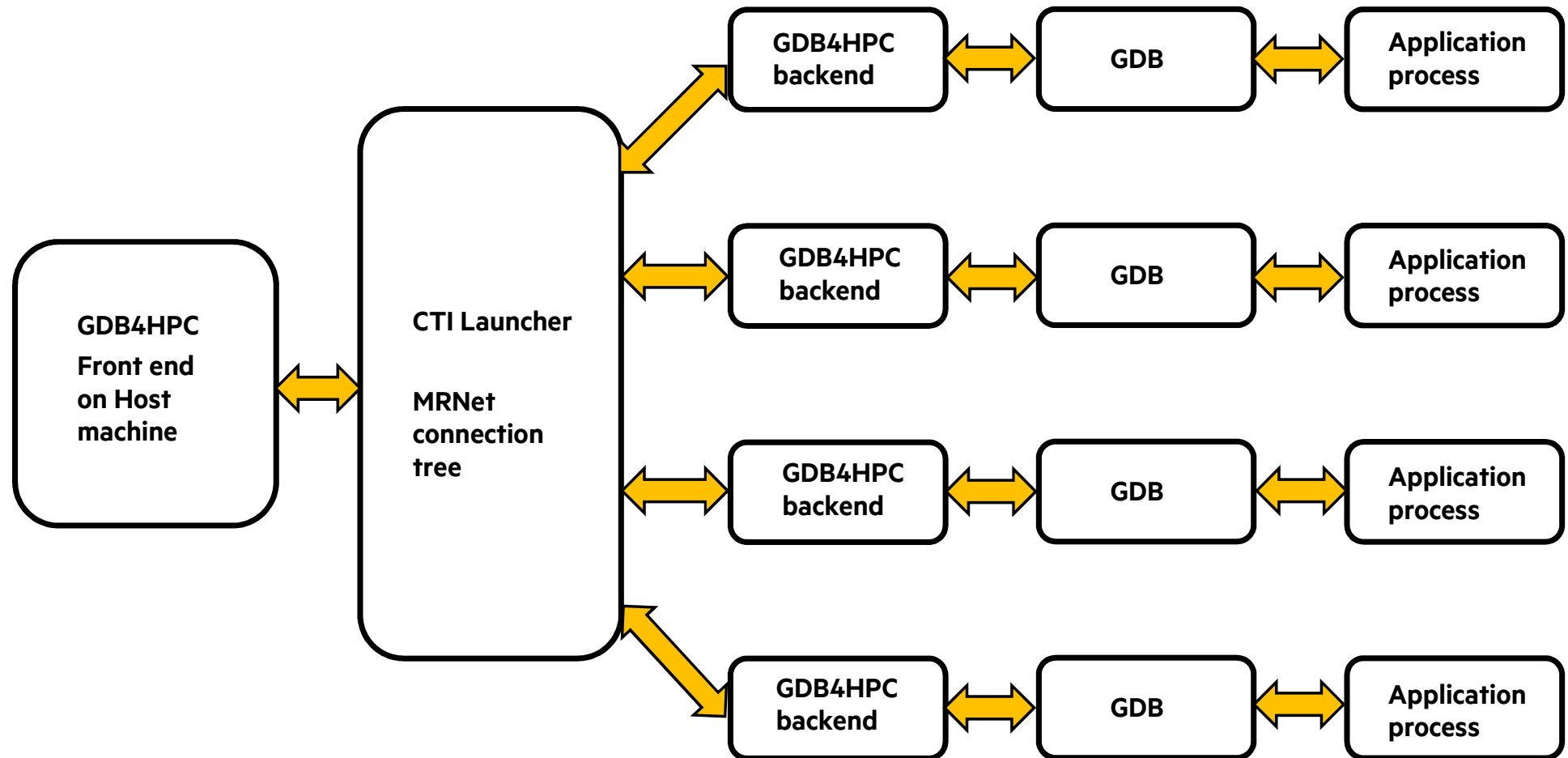
- And probably a lot more

```
dshanks@ln04:/VH1-debug/run> gdb ../bin/vh1-mpi-cray core
GNU gdb (GDB; SUSE Linux Enterprise 15) 10.1
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-suse-linux".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://bugs.opensuse.org/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.
[New LWP 191743]
[New LWP 193054]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Core was generated by `/debugging_workshop/VH1-debug/run/..'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x0000000000413d74 in flatten () at flatten.f90:29
29      do n = nmin-4, nmax+4
[Current thread is 1 (Thread 0x2b4e0268a5c0 (LWP 191743))]
(gdb) bt
#0  0x0000000000413d74 in flatten () at flatten.f90:29
#1  0x0000000000412570 in sweepz () at sweepz.f90:126
#2  0x0000000000000000 in ?? ()
```

GDB Related Support in GDB4HPC

- Supports (almost) all the commonly used gdb commands

- Gdb4hpc isn't a simple forwarder for gdb
  - Must handle all its special HPC related syntax
  - Understand data types to transmit data efficiently
  - Maintains its own state, like breakpoints
  - Does syntax checking and source look up on the client side

- Gives a "gdbmode" command to access any gdb commands it doesn't handle.

- Also, an escape for expression handling
  - Drawback is that gdb4hpc can only treat these as raw text

- No "convenience features", shell control structures and variables, auto completion, etc.

GDB4HPC Architecture

GDB4HPC Front end on Host machine ↔ CTI Launcher / MRNet connection tree

- GDB4HPC backend ↔ GDB ↔ Application process
- GDB4HPC backend ↔ GDB ↔ Application process
- GDB4HPC backend ↔ GDB ↔ Application process
- GDB4HPC backend ↔ GDB ↔ Application process

GDB4HPC Usage: Run a Job

```
> module load gdb4hpc
```

- Load the module to access the gdb4hpc executable.
- Compile the application using either the **-g** or **–Gn** option of the relevant compiler

```
> salloc –N …
> gdb4hpc
dbg all> launch $p{<number of ranks>} --args="<app_args>" app.exe
```

- Launch the application from **gdb4hpc** on a given number of ranks with arguments.
- Do this from an interactive session with enough resources.
- Typical scenario when you want to debug an application from the beginning.

GDB4HPC Usage: Attach to a Running Job

```
> sbatch job.slurm
> module load gdb4hpc
> gdb4hpc
dbg all> attach $p <slurm_jobid>.<slurm_stepid>
```

- **gdb4hpc** can attach to a running application (job step) from the login node
- Get the `<slurm_stepid>` with `sstat --format "JobID" <slurm_jobid>`, eg.

```
>sstat --format "JobID" 2053748
JobID
------------
2053748.0
```

Deadlock Example

- The dollar sign and curly brackets are specific gdb4hpc syntax
  - $p is the process set name in this example
  - Each is identified by a unique process set identifier and the curly braces are used for indexing a process set

  - More typically it specifies the members of a subset, examples
    - $p{4}, $p{0..9}, $p{0,3,7}, $p{0,11..16}
  - `viewset $p` to see members

```
> gdb4hpc
gdb4hpc 4.14.2 - Cray Line Mode Parallel Debugger
With Cray Comparative Debugging Technology.
Copyright 2007-2022 Hewlett Packard Enterprise Development LP.
Copyright 1996-2016 University of Queensland. All Rights Reserved.

Type "help" for a list of commands.
Type "help <cmd>" for detailed help about a command.
dbg all> attach $p 2053796.0
0/2 ranks connected... (timeout in 299 seconds)
2/2 ranks connected.
Created network...
Connected to application...
Current rank location:
p{0}: #0   0x000014bd843b5ee6 in MPIDI_SHMI_progress
p{0}: #1   0x000014bd82e68059 in MPIR_Wait_impl.part.0
p{0}: #2   0x000014bd82901fc0 in PMPI_Recv
p{0}: #3   0x0000000000201b74 in main at
/pfs/lustrep2/projappl/project_465000297/alfiolaz/work/debugging/dea
dlock/deadlock.c:25
p{1}: #0   0x000014561e7961e5 in MPIDI_CRAY_Common_lmt_progress
p{1}: #1   0x000014561e78b599 in MPIDI_SHMI_progress
p{1}: #2   0x000014561d23d059 in MPIR_Wait_impl.part.0
p{1}: #3   0x000014561ccd6fc0 in PMPI_Recv
p{1}: #4   0x0000000000201bc4 in main at
/pfs/lustrep2/projappl/project_465000297/alfiolaz/work/debugging/dea
dlock/deadlock.c:30
```

GDB4HPC Extensions: Process Set

- We can use focus to set context
  - `focus $p{0..3}`
  - `focus $all`
- We can define a new process set with `defset`
  - `defset $few $p{0..3}`
  - `focus $few`
- Can also use sets in variable references, eg. printing a variable
  - `p $few::mycount`

```
p{0..19}: Initial breakpoint, main at /mnt/lustre/a2fs-work2/work/y02/y02/harveyr/examples/pi/C/pi_mpi.c:10
dbg all> l 18,19
p{0..19}: 18     MPI_Comm_rank(MPI_COMM_WORLD,&rank);
p{0..19}: 19     MPI_Comm_size(MPI_COMM_WORLD,&size);
dbg all> break 18
p{0..19}: Breakpoint 1: file /mnt/lustre/a2fs-work2/work/y02/y02/harveyr/examples/pi/C/pi_mpi.c, line 18.
dbg all> cont
p{0..19}: Breakpoint 1, main at /mnt/lustre/a2fs-work2/work/y02/y02/harveyr/examples/pi/C/pi_mpi.c:18
dbg all> focus $p{0..3}
dbg p_temp> next
p{0..3}: main at /mnt/lustre/a2fs-work2/work/y02/y02/harveyr/examples/pi/C/pi_mpi.c:19
dbg p_temp> p rank
p{0}: 0
p{1}: 1
p{2}: 2
p{3}: 3
dbg p_temp> focus $all
dbg all> p rank
p{0,4..19}: 0
p{1}: 1
p{2}: 2
p{3}: 3
```

GDB4HPC with ROCGDB

- Running GDB4HPC on a GPU application
  - launch $p{<number of ranks>} **--gpu** --args="<app_args>" app.exe
  - **--gpu enables the possibility to use rocgdb** for GPU kernels
    - A single tool to debug MPI, CPU threads and GPU kernels
  - add **--env="MPICH_GPU_SUPPORT_ENABLED=1**" to enable **GPU-aware communications**

AN EXAMPLE GDB4HPC LAUNCH

Start the debugger

```
faces-tests> gdb4hpc
gdb4hpc 4.13.10 - Cray Line Mode Parallel Debugger
With Cray Comparative Debugging Technology.
Copyright 2007-2021 Hewlett Packard Enterprise Development LP.
Copyright 1996-2016 University of Queensland. All Rights Reserved.

Type "help" for a list of commands.
Type "help <cmd>" for detailed help about a command.
dbg all> launch $a{16} --gpu --env="MPICH_GPU_SUPPORT_ENABLED=1" -g "-N 2 -p bp11" -i opt.in ./faces
Starting application, please wait...
Creating MRNet communication network...
sbcast: error: No compression library available, compression disabled.
sbcast: error: No compression library available, compression disabled.
Waiting for debug servers to attach to MRNet communications network...
Timeout in 400 seconds. Please wait for the attach to complete.
Number of dbgsrvs connected: [1];   Timeout Counter: [0]
Number of dbgsrvs connected: [1];   Timeout Counter: [1]
Number of dbgsrvs connected: [16];   Timeout Counter: [0]
Finalizing setup...
Launch complete.
a{0..15}: Initial breakpoint, main at /lus/cflus02/sabbott/faces/hip/gpu_subtle/main.cpp:103
dbg all> █
```

Launch command

Thread aggregation in GDB4HPC

```
a{0..15}: Initial breakpoint, main at /lus/cflus02/sabbott/faces/hip/gpu_subtle/main.cpp:103
dbg all> c
<$a>: 0 with node rank 0 using device 0 (8 devices per node) (asked for 0)
<$a>: 8 with node rank 0 using device 0 (8 devices per node) (asked for 0)

dbg all> info thread
a{8}: Debugger error: Gdb get thread info failed.
a{0..5,7,9..10,13}: *** The application is running
a{11..12,14..15}:    Id      Frame
a{11..12,14..15}: * 1-3    "faces" (running)
a{11..12,14..15}:    4-2313 AMDGPU "faces" void gpuRun2x3<Faces::share(DArray<double, 6>&)::{lambda(int, int,
 int, int, int)#1}>(Faces::share(DArray<double, 6>&)::{lambda(int, int, int, int, int)#1}, int, int, int, in
t, int) [clone .kd] () from file:///lus/cflus02/sabbott/faces/hip/gpu_subtle/faces#offset=77824&size=267392
a{11..12,14..15}:
a{6}:    Id      Frame
a{6}: * 1-3    "faces" (running)
a{6}:    4-443 AMDGPU "faces" ?? ()
a{6}:
dbg all> █
```

We're in non-stop mode by default, so some threads halting doesn't necessarily stop everything

gdb4hpc tries its best to aggregate information

(but sometimes aggregation does break down)

In non-stop mode you can halt the threads, e.g. `halt -a` for all threads

40

Final remarks for GDB4HPC

- More information with man page gdb4hpc
  - https://cpe.ext.hpe.com/docs/debugging-tools/index.html#gdb4hpc

- The key concept that gdb4hpc overlays on gdb (rocgdb)
  - A parallel harness and aggregator around gdb, rocgdb
  - Moving on from this you can use gdb to follow execution paths, view the state of variables etc. to trace more insidious bugs in an application

- The tool can be used for investigating hanging or crashed applications
  - Such information is useful if you submit a helpdesk query

Valgrind for HPC

Valgrind-based debugging tool for parallel applications

Valgrind for HPC

```
> module load valgrind4hpc
```

- Load the module to access the valgrind4hpc executable
- Target executables must be built dynamically and contain debug symbols (**-g** option).

```
> salloc …
> valgrind4hpc –n4 --launcher-args="-N2" --valgrind-args="--track-origins=yes --leak-check=full" ./a.out -- arg1 arg2
```

- get an interactive session via **salloc**
- **-n** to specify the number of ranks
- **--launcher-args** to specify other SLURM flags
- **valgrind4hpc** and target program arguments should be separated by two dashes, **--**
- Many suppressions by default:
  /opt/cray/pe/valgrind4hpc/<version>/share/suppressions/

Valgrind for HPC Example

```
19: int *test = (int*)malloc(4*sizeof(int));
20:
21: if(rank==0)
22: {
23:         test[6] = 1;
24:         test[10] = 2;
25: }
```

```
RANKS: <0>

Invalid write of size 4
  at main (in hello.c:23)
Address is 8 bytes after a block of size 16 alloc'd
  at malloc (in vg_replace_malloc.c:306)
  by main (in hello.c:19)
```

- Man page valgrind4hpc for more info
  - https://cpe.ext.hpe.com/docs/debugging-tools/valgrind4hpc.1.html

Sanitizers for HPC

Use several tools to check program correctness at run-time for parallel applications

Perform dynamic analysis of parallel programs with sanitizers4hpc

- Sanitizers4hpc is a debugging tool to aid in the detection of memory leaks and errors in parallel applications
  - Static instrumentation at compile time via `-fsanitize=<sanitizer>`
    - Sanitizers are: Address, Leak, Thread (https://github.com/google/sanitizers)
  - It aggregates any duplicate messages across ranks to help provide an understandable picture of program behavior

- Sanitizers4hpc supports the Sanitizer libraries included with both the Cray CCE and the GNU GCC compilers
  - Cray Fortran only supports Address and Thread sanitizers
  - Note: Address Sanitizer and Thread Sanitizer cannot be used simultaneously

- More info: `man sanitizers4hpc`
  - https://cpe.ext.hpe.com/docs/debugging-tools/sanitizers4hpc.1.html

Sanitizers for HPC

- Compile the application with `-f sanitize=<sanitizer>`, e.g.

```
cc –g -fsanitize=leak leak.c -o leak
```

- Load the module to access the sanitizers4hpc executable

- Get an interactive session via salloc

```
> salloc …
> sanitizers4hpc -l "-n4" -- ./a.out arg1 arg2
```

- `-l` to pass arguments to the system launcher (e.g. Slurm)

- The target binary and its arguments are listed after the double dash -

- More info and examples: `man sanitizers4hpc`

Sanitizers for HPC Example: Address sanitizer

```
 1: program address
 2:    implicit none
 3:
 4:    integer, dimension(10) :: array
 5:
 6:    array = 2
 7:
 8:    array(12) = 3
 9:
10:   print *, array
11:
12: end program address
```

```
RANKS: <0>
AddressSanitizer: global-buffer-overflow on address 0x000000551eec at pc 0x00000041f95c bp 0x7ffd3da8b890 sp
0x7ffd3da8b888
WRITE of size 4 at 0x000000551eec thread T0
    #0 0x41f95b in address_ /home/users/alazzaro/lumi_coe/sanitizers/fortran/address.f90:8
    #1 0x7f8ba5aad2bc in __libc_start_main (/lib64/libc.so.6+0x352bc) (BuildId:
28910b266cdd8f0c54c7830b758e4a1339f255c1)
    #2 0x41f429 in _start /home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x86_64/start.S:120
```

CRAY_ACC_DEBUG

For debugging GPU applications "for free"

CRAY_ACC_DEBUG

```
> CRAY_ACC_DEBUG=<1,2 or 3> srun -n … ./exe …
```

- Only for CCE and OpenACC/OpenMP offload
- AMD has an equivalent mechanism for HIP code
- Provides a mechanism for all offloading operations
- For "free"
  - Do not need to recompile
  - Do not need to add flag for compiling or linking
- Set value gives an increasing level of verbosity
  - 1 : very low level (not enough for debugging)
  - 2 : recommended for users
  - 3 : recommended for experts

CRAY_ACC_DEBUG: outputs

- CRAY_ACC_DEBUG=1
  - Transfers between host and accelerator
    - Number of items transferred but not the name of the transferred data
    - Line numbers
  - Kernel executions

- CRAY_ACC_DEBUG=2 same as =1 +
  - Transfers between host and accelerator
    - Number of items transferred and the name of the transferred data
    - Data already present on the device
    - Variables on the map directives are listed
    - Arrays with unknown shape information at compile time shown with question marks

- CRAY_ACC_DEBUG=3 same as =2 + more (ex: memory found or not in present table...)

```
CRAY_ACC_DEBUG=2 srun -N 1 -n 1 -p amdMI100 ./omp_exe
ACC: Version 4.0 of HIP already initialized, runtime version 3212
ACC: Get Device 0
ACC: Set Thread Context
ACC: Start transfer 1 items from main.f:17
ACC: allocate, copy to acc 'op_ptr' (120 bytes)
ACC: End transfer (to acc 120 bytes, to host 0 bytes)
ACC: Start transfer 3 items from main.f:17
ACC: allocate, copy to acc 'op_ptr%array(?:?,?:?)' (80000 bytes)
ACC: present 'op_ptr' (120 bytes)
ACC: attach pointer 'op_ptr%array' (96 bytes)
ACC: End transfer (to acc 80000 bytes, to host 0 bytes)
ACC: Start transfer 1 items from main.f:17
ACC: allocate, copy to acc 'op_ptr_b' (128 bytes)
ACC: End transfer (to acc 128 bytes, to host 0 bytes)
ACC: Start transfer 3 items from main.f:17
ACC: allocate, copy to acc 'op_ptr_b%base_type%array(?:?,?:?)'
(80000 bytes)
ACC: present 'op_ptr_b' (128 bytes)
ACC: attach pointer 'op_ptr_b%base_type%array' (96 bytes)
ACC: End transfer (to acc 80000 bytes, to host 0 bytes)
```

Questions?