`minitest`: Framework for Testing A GPU Performance Tool

Marty Itzkowitz, Software Consultant

`itzkowitzmarty@gmail.com`

HPCToolkit Group, Rice University

Scalable Tools Workshop

Granlibakken, Lake Tahoe, California

June 19, 2023

Program Agenda

- Introduction
- RunLists and the `minitest` TargetApp's
- TestLists and The DataDescriptor
- How it Works
- Conclusion

Introduction, I

- Design goal: build a framework for testing HPCToolkit CPU and GPU profiling
  - Needed to cope with rapidly evolving Vendor GPU SW stacks and APIs
  - Needed to cope with each site's OS peculiarities
  - Needed to cope with multiple compilers and their quirks
- Invoked by "**`minitest -r`** <RunList> <TestList>"
  - <RunList> is a list of directories, each of which builds a TargetApp
  - <TestList> is a list of tests to be run in each directory against that TargetApp
  - Runs a double loop:
    - **`cd`** to each directory in the <RunList>; run each test in the <TestList>
    - Each test writes TESTPASS or TESTFAIL to the log
    - Reports "SUCCESS" (if 0 TESTFAIL's in log) or "FAILED" (if >0 TESTFAIL's in log)

Introduction, II

- Alas, the compiler modules necessary for different GPUs are incompatible
  - We can't do a single run for all predefined RunLists
- The **`QA.minitest`** script does multiple **`minitest`** runs
  - It always runs the CPU tests
  - It runs the GPU tests for each GPU flavor found on the system
    - Loading and unloading the compiler modules
  - It will typically need tweaking for each machine's module structure
- The **`sum.minitest`** script summarizes the run(s)
  - Can summarize completed run or run in progress

Introduction, III

Summary of a **`QA.minitest`** run, which took about 15 minutes

```
  Summary of minitest -r cpu full run: (this run) tests passed: 114; tests failed: 0; tests skipped: 30
  Summary of minitest -r cuda full run: (cum.ulative) tests passed: 291; tests failed: 0; tests skipped: 45
  Summary of minitest -r rocm full run: (cum.ulative) tests passed: 425; tests failed: 4; tests skipped: 60

    Summary of minitest run as of Fri May  5 20:11:26 CDT 2023
  Total tests: 489;  passing tests = 425; failing tests = 4 ; skipped tests = 60

Summary of data collection options; tests with multiple options are counted for each option
    run-only tests: 42;   passing tests = 40; failing tests = 2
     CPUTIME tests: 207;  passing tests = 193; failing tests = 2; skipped tests = 12
    REALTIME tests: 81;   passing tests = 81; failing tests = 0; skipped tests = 0
      cycles tests: 225;  passing tests = 165; failing tests = 0; skipped tests = 60
       insts tests: 60;   passing tests = 60; failing tests = 0; skipped tests = 0
        PAPI tests: 60;   passing tests = 0; failing tests = 0; skipped tests = 60

       nvgpu tests: 72;   passing tests = 72; failing tests = 0; skipped tests = 0
     nvgpupc tests: 48;   passing tests = 48; failing tests = 0 skipped tests = 0
      amdgpu tests: 63;   passing tests = 61; failing tests = 2; skipped tests = 0

Summary of failure modes
HSA STATUS ERROR OUT OF RESOURCES failures: 2 of 4 total failures
               timeout triggered failures: 2 of 4 total failures
```

Program Agenda

- Introduction
- RunLists and the **`minitest`** TargetApp's
- Testlists and the DataDescriptor
- How it Works
- Conclusion

RunLists and the **`minitest`** TargetApp's, I

- A <RunList> is a file with a list of target directories
  - Predefined <RunList>'s: **`cpu`**, **`cuda`**, **`level0`**, **`rocm`**
  - Any other name is assumed to be a file with the user's custom RunList
- All target directories are inside directory **`.../minitest/`**<subdir>/
  - <subdir>/ is one of **`cpu/`**, **`amdgpu/`**, **`intelgpu/`**, or **`nvidiagpu/`**
- TargetApps combine <front-end> + <back-end>, built by a <compiler>
  - Target directories are named <front-end>.<back-end>.<compiler>
  - Target directory contains only a **`Makefile`** to build the TargetApp
  - TargetApp is named <front-end>.<back-end>.<compiler>.<gputype>

RunLists and the **`minitest`** TargetApp's, II

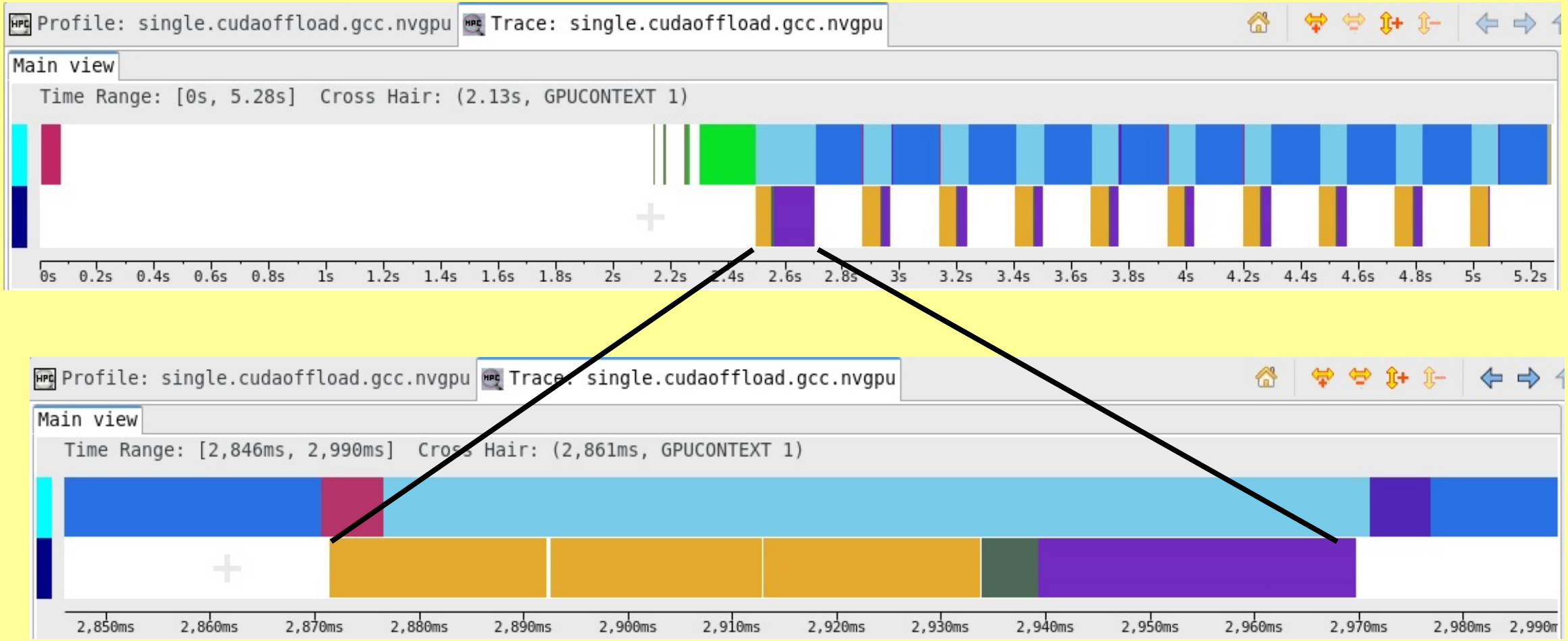
- Front-ends are:
  - **`ompthreads.cc`**, **`posixthreads.cc`**, **`single.cc`**
- Back-ends are:
  - **`nooffload.cc`**, **`ompgpu.cc`** (OpenMP offload)
  - Vendor-specific offloading:
    - **`cudagpu.cu`**, **`hipgpu.hip.cpp`**, and **`syclgpu.cc`**
  - All **`#include`** a common **`compute.h`**
  - Defines the actual computation
    - Ensures identical computation in all Target Apps

RunLists and the `minitest` TargetApp's, III

- All the TargetApp's behave the same way:
  - The Front-end:
    - Allocate and initialize 3 arrays of size *N* (default 40000000) of doubles for each thread
    - Spawn the worker threads (or become a worker thread)
    - Reap the worker threads when they are done
    - Validate the results
  - Each worker thread:
    - Iterate *N* times (default 3), calling **`twork()`** and then **`spacer()`**
  - The Back-end implements **`twork()`** to offload the computation (or not)
    - Copy the 3 arrays to GPU, spawn the Kernel, copy third array back
- Behavior makes the trace easy to understand

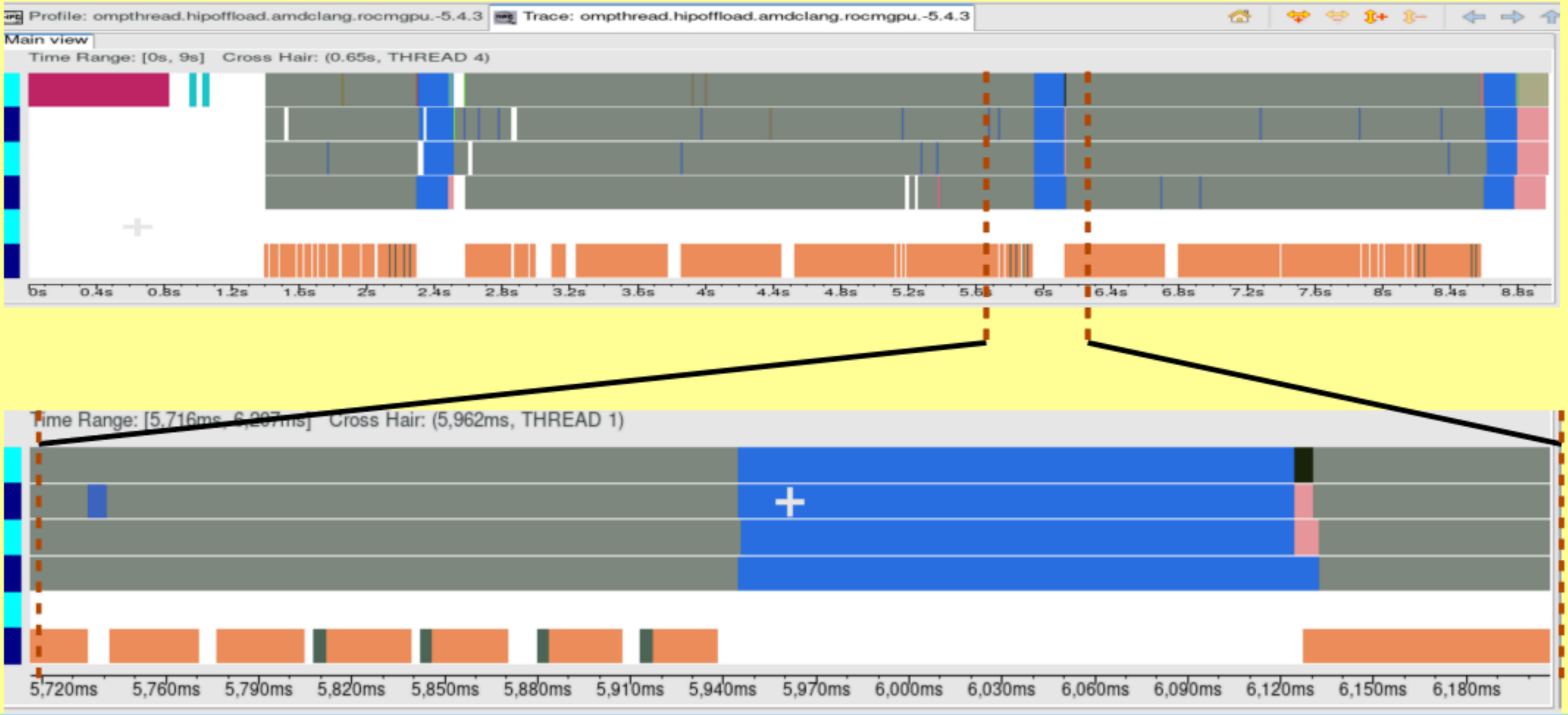
RunLists and the `minitest` TargetApp's, IV

Screen shot of single-threaded run, Nvidia GPU, cuda-offload, 10 iterations
Zoom in on first iteration: three copy-in's, then the kernel launch, then the copy-out



RunLists and the `minitest` TargetApp's, V

Screen shot of four-threaded run, AMD GPU, hip-offload
Zoom in on an interesting region



Program Agenda

- Introduction
- RunLists and the `minitest` TargetApp's
- TestLists and the DataDescriptor
- How it Works
- Conclusion

TestLists and the DataDescriptor, I

- <TestList> is a file containing a list of tests
- Each test is defined by a DataDescriptor
  - DataDescriptor is of the form **`expt.*`** or **`run.*`** (the **`*`** is explained below)
- Predefined <TestList>'s are: **smoke** and **full**
  - They each have variants for each predefined <RunList>. *i.e.*, each GPU type
  - **smoke** runs a few tests in each directory
  - **full** runs many tests in each directory
    - A third predefined TestList is **stress**, not recommended for user use
    - **stress** runs many tests with very high frequency profiling in each directory
- Any other <TestList> argument is a user file with a custom set of tests

TestLists and the DataDescriptor, II

- The DataDescriptor is either
  - **`expt.dt1.dt2.dt3....dtN`**
    - To run the TargetApp under **`hpcrun`** and process the data; or
  - **`run.dt1.dt2.dt3....dtN`**
    - To run the TargetApp without any data collection
- The various **`.dti.`** elements are referred to as “data tags”
  - Some correspond to data collection arguments:
    - **`.cputime.`**, **`.realtime.`**, **`.cycles.`**, **`.insts.`**, **`.papicycles.`**, **`.insts.`**, **`.t.`**
    - **`.nvgpu.`**, **`.nvgpupc.`**, **`.amdgpu.`**, **`.level0gpu.`**
  - Others correspond to run-time options to the TargetApp:
    - **`.NN.`**, **`.tracker.`**, **`.MI.`**, **`.MN.`**
      - Those last two are specific options to minitest TargetApps for iteration count and array size.
    - **`.-<user-label>`** is an arbitrary user-specified string (must be last data tag)

TestLists and the DataDescriptor, III

Some Sample DataDescriptors and their Meaning

`expt.2.cputime.cycles.insts.t`

Run 2 worker threads, collect profile data for CPU Time, cycles, and instructions, with trace data

`run.1.MI,10.tracker`

Run 1 worker thread for ten iterations, collect no data, and simulate behavior of LLNL's tracker

(The latter is a barn-door lock implemented when the real tracker broke HPCToolkit)

`expt.1.realtime.nvgpu.t`

Run 1 worker thread, collect profile data for Real Time and Nvidia GPU data, with trace data

`expt.4.cputime.amdgpu.t`

Run 4 worker threads, collect profile data for CPU Time and AMD GPU data, with trace data

TestLists and the DataDescriptor, IV

- The DataDescriptor string is appended to all directory and file names
  - The measurement directory: **`meas.DataDescriptor`**
  - The database directory: **`dbase.DataDescriptor`**
  - A logfile of the experiment or run: **`log.DataDescriptor`**
    - Makes it easy to identify files corresponding to an experiment or run
  - Allows many experiments with different DataDescriptors in a directory
  - Allows repeated experiments in a directory, varying the **`.-<user-label>`**

Program Agenda

- Introduction
- RunLists and the `minitest` TargetApp's
- TestLists and the DataDescriptor
- How it Works
- Conclusion

How it Works, I

- A tangled web we weave, ...
  - Reflects the incremental accretion of functionality; but

    If it ain't broke, don't fix it (at least not now)
- The **`minitest`** script is invoked with a <RunList> and a <TestList>:
  - Keeps a **`log.minitest`** file for all operations
  - Loops over the directories in the <RunList>
    - Runs "**`make`** <TestList>" in each directory
    - That invokes one of the **`runsuite.*`** scripts
      - Choice of which such script depends on the **`Makefile`** in the directory
      - Choice also depends on the <gputype> in the TargetApp

How it Works, II

- The **`runsuite.*`** scripts:
  - Write a **`LOG.suite.*`** file for all operations
  - Read the <TestList> file
  - Loop over the tests in the file:
    - Invoke **`dohpct`** on the TargetApp and args, passing in the DataDescriptor for the test

- **`runsuite.cuda`** and **`runsuite.rocm`** also support multiple vendor versions:
  - Input a list of **`cuda`** and **`rocm`** versions, respectively; loop over the versions in the list
    - Unload current module, load the module for that version
    - Run the <TestList> suite, appending the **`cuda`**/**`rocm`** version string to all names

How it Works, III

- The **`dohpct`** command:
  - Invoked with two arguments: “TargetApp args” and the DataDescriptor
  - Parses the DataDescriptor
    - Maintaining list of prepend commands
    - Maintaining list of arguments for **`hpcrun`**
    - Maintaining list of arguments to **`hpcstruct`**, as implied by **`hpcrun`** argument
  - Formats a shell command
    - Starts with the prepend commands
    - If Data Descriptor starts with **`run.`**, adds a **`runrun`** command and args
    - If Data Descriptor starts with **`expt.`**, adds a **`runhpct`** command and args
  - Invokes **`system`**(command), thus executing either **`runrun`** or **`runhpct`**

How it Works, IV

- The **`runrun`** script:
  - Is invoked with two arguments: “TargetApp args”, DataDescriptor
  - Formats a shell command to run the TargetApp with its arguments
  - Launches the shell command, under **`timeout`** and **`/bin/time`**
  - Examines the exit code, output files, *etc.*, to look for possible failure modes
  - Finishes by writing a TESTPASS or TESTFAIL line to the master **`log.minitest`** file
    - The name of the individual **`log.DataDescriptor`** file is always inserted
      - Makes it easy to cut-and-paste to see the details of the run, successful or not
      - If TESTFAIL, the failure mode is also inserted into that line

How it Works, V

- The **`runhpct`** script:
  - Is invoked with four arguments: “TargetApp args”, **`hpcrun`** args, DataDescriptor, **`hpcstruct`** args
  - Formats a shell command to run **`hpcrun`** with the **`hpcrun`** args on the TargetApp and its arguments
  - Launches the shell command, prefaced by **`timeout`** and **`/bin/time`**
    - Examines the exit code, output files, *etc.*, to look for possible failure modes in data collection
  - If no failures are noted, invokes **`hpcstruct`** with its args on the measurements directory
    - Examines the exit code, output files, *etc.*, to look for possible failure modes in **`hpcstruct`**
  - If no failures are noted, invokes **`hpcprof`** to create the database directory
    - Examines the exit code, output files, *etc.*, to look for possible failure modes in that step
  - Finally, writes TESTPASS or TESTFAIL to the master **`log.minitest`** file
    - Contains the path to **`log.DataDescriptor`** file
      - Makes it easy to copy-and-paste to see details of the run
    - If TESTFAIL, the failure mode is also inserted into that line

How it Works, VI

- Future Plans
  - Add Fortran versions of the Test Directories and TargetApp's
  - Add MPI versions of the Test Directories
  - Develop simple configuration management scheme
    - To determine compilers, GPU SW versions, module paths, *etc.*
  - Implement scheme to validate recorded data
    - Ensure that the data base reflects the real program behavior
  - Untangle the architectural web (perhaps)

Program Agenda

- Introduction
- RunLists and the `minitest` TargetApp's
- Testlists and the Data Descriptor
- How it Works
- Conclusion

Conclusion

- **`minitest`** has met its design goals
  - It has tested HPCToolkit on many sites and compilers
    - Different OS versions, different OpenMP implementations, different vendor stacks
      - With multiple CUDA and ROCM versions installed
  - It has uncovered bugs in:
    - HPCToolkit; GPU SW; compilers; **`libmonitor`**
      - Some are fixed, some not; some are not yet understood
      - HPCToolkit wrapping of vendor OpenMP GPU SW caused failure
      - **libmonitor** failed to initialize before GPU Vendor runtime
      - HPCToolkit failures do handle unannounced Vendor ABI changes
  - It has revealed idiosyncrasies in various sites' environments
    - An glibc library version that did not return from **`fork()`**
    - A Vendor GPU driver that crashed node when running minitest
    - LLNL's **`tracker`**, an application from an execute-only file

For More Information, Download the Repository

- **`minitest`** is a ***SMALL*** GitLab repository

  `https://gitlab.com/hpctoolkit/minitest.git`

  - Sources: **`.../minitest/src/*.{c,.cc,.cpp,.h,.cu}`**
    - 12 files, totalling ~2500 lines
  - Scripts: **`.../minitest/bin/{*run*,*minitest}`**
    - 10 files, totalling ~ 2500 lines
  - RunLists, TestLists, Makefiles, *etc.* are all quite small
- **`minitest`** is specific to HPCToolkit
  - However, it would be relatively easy to port to another performance tookit:
    - **`runhpct`** script needs to change for the other toolkit's workflow, commands, and error output
    - The DataDescriptor needs to change for new/different options to the other toolkit's commands
    - **`dohpct.c`** needs to change to parse the new DataDescriptor