Getting feedback at EuroHack

Alistair Hart
ahart@cray.com
(plus help from PGI and Nvidia folk)

© Cray Inc.

What's going on?

- **You need feedback to see what is happening**
- **Two forms of feedback available**
  - Compiler feedback
    - Static, says what the compiler intends to do
  - Runtime feedback
    - Dynamic, says what the runtime actually did

- **Compiler feedback is free**
  - No performance overhead
  - But you need to ask for it
  - You should always ask for it

- **Runtime feedback has a performance overhead**
  - Use it when developing, not for performance testing or production
  - CrayPAT, nvprof are the "Rolls-Royce" solutions
  - There are also some less powerful methods, described here

© Cray Inc.

Compiler feedback

- **Cray compiler:**
  - `-hlist=a`
  - For every source file (foo.f, foo.c), get a new file when compile: foo.lst
  - Lots of information about what compiler did (or didn't do)
- **PGI compiler:**
  - `-Minfo=accel`
  - Information written to STDOUT when you compile

© Cray Inc.

Quick runtime feedback

- **A really quick way to see what is happening with your code as it runs on the GPU (or GPUs)**
- **It's not scalable**
  - There is a lot of information
    - Commentary: Event-by-event, ball-by-ball, blow-by-blow
  - The longer your code runs, the more information there is
    - Multiplied by the number of MPI ranks
  - It will slow down code execution
    - And probably skew the profile slightly
- **Three quick methods:**
  - Each enabled by environment variable at runtime (in jobscript)
    - No need to recompile
  - Each gives text output
  - Don't use more than one at once!

© Cray Inc.

Nvidia Compute Profiler (PGI, Cray, CUDA)

- `export COMPUTE_PROFILE=1`
- **Gives timing information for each event**
  - Data transfers
  - Kernel executions
  - Written to a new text file (you can specify the name)
  - Works for PGI and Cray compilers and CUDA
- **Very useful to get some quick profiling**
  - See how much time is spent in computation vs. data transfers
- **Tip from Nvidia:**
  - To integrate Compute Profiler output with the application output:
    - `export COMPUTE_PROFILE_LOG=/dev/stdout`

© Cray Inc.

CRAY_ACC_DEBUG (just Cray)

- `export CRAY_ACC_DEBUG=1`, `2` or `3`
  - Recommend level 2

- **Gives array movement information**
  - Name of the arrays
  - Number of bytes transferred
  - Written to STDERR
  - Just for the Cray compiler
  - Has an API to restrict when information is listed
    - Fortran example on next slide.
    - For C/C++ and more details, see: `man openacc`)

- **Very useful to understand data movements**
  - What takes the time, debugging correctness errors

© Cray Inc.

CRAY_ACC_DEBUG API (Fortran example)

```
! Execute code with CRAY_ACC_DEBUG=1, 2 or 3 in jobscript
PROGRAM main

USE openacc_lib                          ! exposes the API calls
INTEGER :: cray_acc_debug_orig           ! preserve original value

<start of executable code>
cray_acc_debug_orig = cray_acc_get_debug_global_level()
CALL cray_acc_set_debug_global_level(0)

<code without commentary>

CALL cray_acc_set_debug_global_level(cray_acc_debug_orig)
<code with commentary>
CALL cray_acc_set_debug_global_level(0)

<code without commentary>

END PROGRAM main
```


© Cray Inc.

PGI_ACC_NOTIFY (just PGI)

- **`export PGI_ACC_NOTIFY=1, 3, 7, 15, 31`**
  - Recommend level 3 (kernel launches, data movement)

- **Gives array movement information**
  - Name of the arrays
  - Number of bytes transferred
  - Written to STDERR
  - Just for the PGI compiler

- **Very useful to understand data movements**
  - What takes the time, debugging correctness errors

- **`export PGI_ACC_TIME=1`**
  - gives a summarised output for whole program
  - probably shouldn't do this at the same time as `PGI_ACC_NOTIFY`

© Cray Inc.

MPI programs

- The problem is that all the information from each rank comes out at once, and gets mixed up together
- Better to separate the information to one file per rank
- There is a trick to do this in your jobscript
  - Rather than:
    - `aprun <aprun_options> <EXE> <EXE options>`
  - We now use:
    - `aprun <aprun_options> bash wrapper.bash <EXE> <EXE options>`
  - `wrapper.bash` is shown on the next slide

© Cray Inc.

wrapper.bash

```
#!/bin/bash
# ONLY ACTIVATE ONE RUNTIME COLLECTION METHOD AT A TIME!!!
# A name for the files (replace FOO as appropriate)
jobstem=$(printf "FOO.%03d" $ALPS_APP_PE)
# NVIDIA COMPUTE_PROFILER: set this 1 to activate
export COMPUTE_PROFILE=0
#   Collect output in separate files, one per process
export COMPUTE_PROFILE_LOG=./${jobstem}.compprof
#   Tune what is collected (optional)
export COMPUTE_PROFILE_CONFIG=compute_profile_config
# Collect CCE runtime information: set this 1,2,3 to activate
export CRAY_ACC_DEBUG=0
# Collect PGI runtime information: set this 1,3 etc. to activate
export PGI_ACC_NOTIFY=0
# Now execute binary with appropriate options
#   Pipe STDERR to separate files
#   (to catch CRAY_ACC_DEBUG, PGI_ACC_NOTIFY commentary)
exec $* 2> ${jobstem}.err
# EOF
```


© Cray Inc.

compute_profile_config

```
# compute_profile_config
method
gputime
cputime
occupancy
memtransfersize
```

- Using file **compute_profile_config** is optional
- It lets you tune what information is collected

© Cray Inc.