Application Debugging and Performance Tuning for Itanium®

Carl Burch, Dave Babcock
WDB Team, HP Caliper Team
Hewlett-Packard

Application Debugging for Itanium: WDB

Exercise 1

- Change directory to ex1
- Issue make
- Start gdb on a.out
- Run to main()
- Continue to marker1()
- Issue next till exit from marker1()
- Step over call to marker2 ()
- Step into marker3 ()
- Use help to understand list command
- Line the source lines for main
- Place a breakpoint at line# 41, and continue
- Print the value of retval
- Continue till program exits

Exercise 2

- Change directory to ex2; make; start gdb on a.out
- Run to main ()
- Place a watchpoint at retval
- Place a temporary breakpoint at marker1 () and continue
- Return out of marker1 () without excuting the remaining part of marker1 ()
- Step over call to marker2 ()
- Step into marker3 ()
- Check the stack trace
- Set a breakpoint at return from factorial ()
- Finish executing marker3 () – use finish command
- Jump over call to marker4 () and continue
- Do a backtrace
- Go up 2 frames and print the variable ‘value’
- Go down 1 frame and print ‘value’
- Use info break command to get the breakpoint information and delete the breakpoint at return from factorial ()
- Continue the program till the watchpoint is hit
- Continue till the program exits

Exercise 3 - threads

- This program creates 3 threads and joins them back in the function do_pass. Each thread executes the routine spin()
- Change directory to ex3/threads; make; start gdb on a.out
- Run to do_pass()
- Break after each thread is spawned
- Continue 3 times and check the thread list after each thread is created
- Switch the current thread to thread 4
- Place a temporary breakpoint at line 56 in thread 4
- Continue and check the thread list
- Apply ‘backtrace’ command to all threads – use thread apply command
- Continue till all the threads join – (line 116)
- Check the thread list now

Exercise 3 – Debugging a running process

- Change the directory to ex3/attach; make
- Start “a.out 1&”
- Attach gdb to this pid
- Get the stack trace and thread info
- Set the variable “wait_here” to 0 to let the program continue beyong the while loop.
- Set a breakpoint at printf and continue the process
- Breakpoint is never hit!!!
- Redo the above steps after loading the shared libraries private
  - Chatr +dbg enable a.out

Exercise 3 – Corefile debugging

- Change directory to ex3/corefiles; make; start gdb on a.out
- Run the program till gdb reports SIGBUS
- Continue to get SIGABRT
- Get the backtrace and continue this program gets terminated
- Restart gdb on a.out with this corefile ‘core’
- Get the backtrace

Exercise 4 – src-no-g

- Change directory to ex4; make; start gdb (no executable)
- Set src-no-g to no_sys_libs
- Read in the executable (file a.out)
- Place breakpoints at main and bar (deferred break)
- Run the program and see that you see source lines
- Continue to bar()
- Finish out of bar()
- Print the value of local *i*
- Continue till program exits

Exercise 4 – Assembly mode debugging

- Rerun the previous program; hits the breakpoint at main
- Disassemble main, and get the first instruction after call to bar()
- Place a breakpoint at this instruction (break *0xaddress)
- Disassemble bar() and get the first instruction of return statement.
- Place a breakpoint at this instruction
- Continue to the breakpoint at bar()
- Continue to the breakpoint at return statement
- Single step (stepi) to next instruction
- Continue to next breakpoint in main
- Print the value of return register r8 (p $r8)

Exercise 5

- Change directory to ex5; make; start gdb on a.out
- enable heap-checking
- run to main
- run to exit (__exit_handler)
- find memory leaks in this program

- Rerun to main
- change frame-count
- change min-leak-size
- run to exit
- find leaks

- Restart gdb; enable heap-checking; run to foo
- print the heap profile
- Enable free checking
- Continue to catch the double free
- enable bounds check
- Continue to catch write beyond bounds

Application Performance Tuning for Itanium: HP Caliper

Making Measurements

caliper <measurement> [options] application ….

ie:

```
caliper cgprof sweep3d
caliper dcache_miss -o report crafty < crafty.in
caliper func_cover -o report --process=all \
        cc -o hello +O hello.c
```

Measurements

		Intrusion
Totals:	cpu_metrics, total_cpu	<< 1%
Profiles:	alat_miss, branch_prediction, dcache_miss, dtlb_miss, fprof, icache_miss, itlb_miss	~ 1% - 3%
Traces:	pmu_trace	
Coverage:	func_cover*	
Counts:	arc_count*, func_count*	~ 10% - 60%
Call graph:	cgprof*	

* not in first Linux release

Demo Applications

- ls -- system directory list utility

  ls

- crafty -- chess program (C)

  crafty < crafty.in

- sweep3d -- pipelined wavefront with line recursion (F90, C)

  sweep3d

- cc -- C compilation (multiprocess)

  cc -o hello +O hello.c

Exercise 1a – getting started

- measure runtime of app:

  caliper total_cpu app

- examine total_cpu report
- measure cpu profile of app:

  caliper fprof -o REPORT app

- explore cpu profile report

Exercise 1b – getting help

- access caliper manpage:

  man caliper

- access caliper help:

  caliper -h

- access caliper info:

  caliper info –r dcache_miss

  caliper info CYCLES

Exercise 2 – measurement options

- measure total fp metrics:

```
caliper  total_cpu  --metric=FP_OPS_RETIRED, \
                              FP_FLUSH_TO_ZERO,  \
                              FP_TRUE_SIRSTALL  \
            app
```

- measure custom total metrics:

```
caliper  my_metrics  app
```

Exercise 3a – report options

- measure dcache misses:

  caliper dcache_miss -d DF app

- generate default dcache miss report:

  caliper report -d DF -o R1

- generate full source dcache miss report:

  caliper report -d DF -o R2 ---context=100

- explore R1 & R2

Exercise 3b – more report options

- try various report options and explore results:

--csv=CSV

--html=HTML

--context-lines=COUNT_SOURCE[,COUNT_DISASSEMBLY]

--detail-cutoff=PERCENT_CUTOFF[,CUMUL_PERCENT_CUTOFF
[,MIN_COUNT]]

--percent-columns=total|cumulative|total:cumulative

--report-details=statement|instruction|statement:instruction

--sort-by=sampled-misses|latency|avg-latency

--summary-
cutoff=PERCENT_CUTOFF[,CUMUL_PERCENT_CUTOFF
[,MIN_COUNT]]

Exercise 4 – measurement context

- measure full application:

```
caliper  fprof  -o R1  app
```

- measure only main module (no shared libraries):

```
caliper  fprof  -o R2  --module-default=none  \
                       --module-include=app  \
                       app
```

- explore R1 & R2

Exercise 5 – a multiprocess app

- measure only root process:

```
caliper  fprof  -o R1  app
```

- measure all processes:

```
caliper  fprof  -o R2  --process=all  app
```

- measure selective processes:

```
caliper  fprof  -o R3  --process=app  app
```

- explore R1, R2 & R3


invent

HP WORLD 2004

Solutions and Technology Conference & Expo

Co-produced by:

interex

shared knowledge • shared power

encompass

AN HP USER GROUP

RECOMMENDED TRAINING VENUE FOR THE

HP Certified Professional