

Performance Analysis Tools: DCPI

Niraj Srivastava, Ph.D.
HPC Expertise Center
HP
niraj.srivastava@hp.com

Performance Profiling with DCPI

Digital Continuous Profiling Infrastructure

DCPI Profiling Tool Suite

From our research labs (SRC, WRL, CRL)

System and/or application profiler

Profiler based on statistical sampling

- Uses CPU performance counters
- Efficient
- Complete: profiles whole system

Analysis tools display collected profiles

- Varying levels of detail
 - image, procedure, instruction

Goals: Identify

- Where the cycles went
- Where peak performance was lost
 - Why?
 - How much?

DCPI Goals

- Accurate, fine-grain profiles
 - $731 \text{ Mhz} / 64\text{K} = \sim 12\text{k samples} / \text{second}$
- Comprehensive: profiles practically all code
- Transparent: no code modification necessary
- Efficient: 1-4 percent overhead
- Compact on-disk storage of sample data
- Continuous use on production systems
- Intended for programmers and optimization tools (e.g. compiler feedback, spike)
- Easy to install, easy to use

System-Wide Information

Total samples for event type cycles = 6095201

The counts given below are the number of samples for each listed event type.

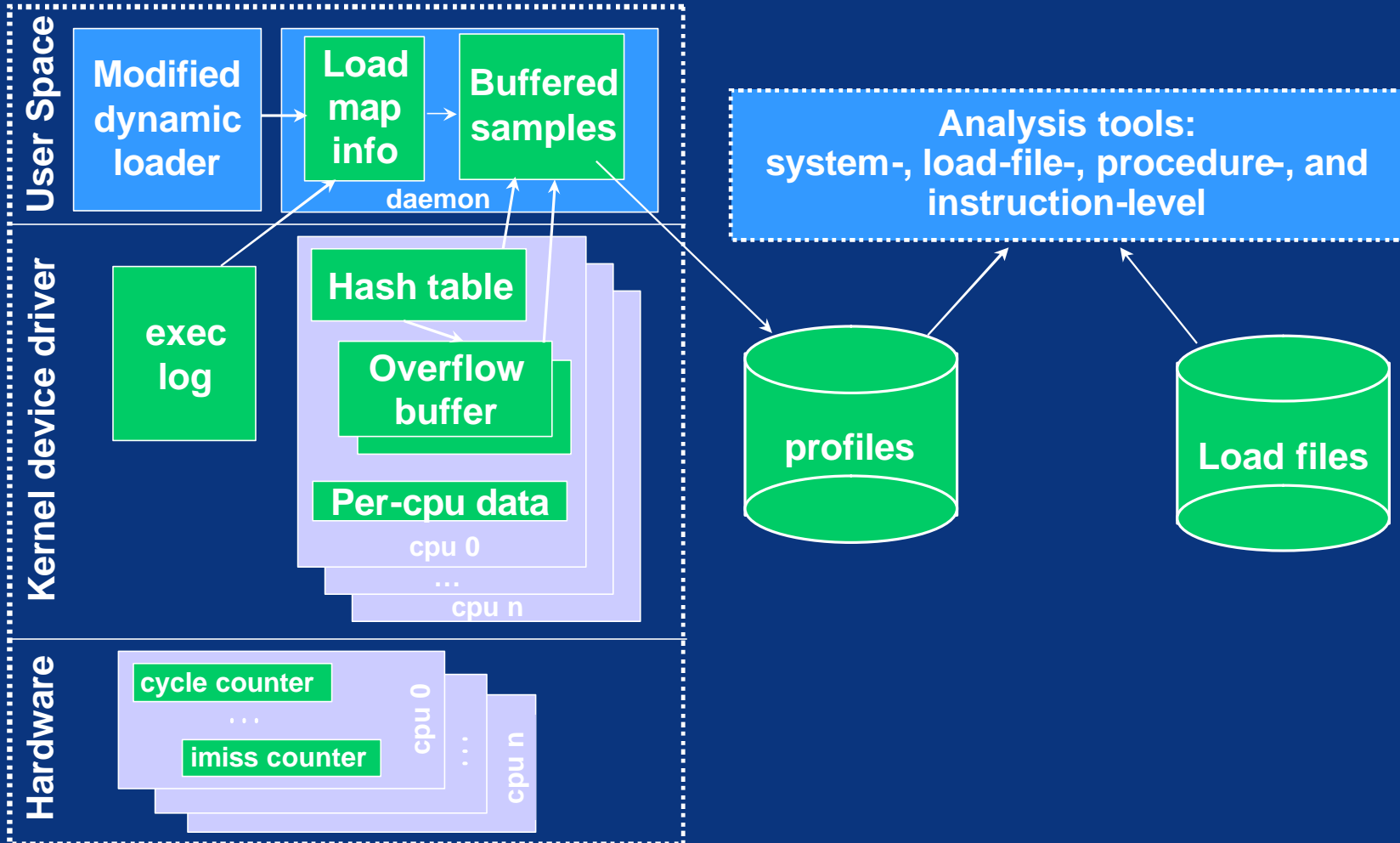
```
=====
```

cycles	%	cum%	procedure	image
2064143	33.87%	33.87%	ffb8ZeroPolyArc	/usr/shlib/X11/lib_dec_ffb_ev5.so
517464	8.49%	42.35%	ReadRequestFromClient	/usr/shlib/X11/libos.so
305072	5.01%	47.36%	miCreateETandAET	/usr/shlib/X11/libmi.so
271158	4.45%	51.81%	miZeroArcSetup	/usr/shlib/X11/libmi.so
245450	4.03%	55.84%	bcopy	/vmunix
209835	3.44%	59.28%	Dispatch	/usr/shlib/X11/libdix.so
186413	3.06%	62.34%	ffb8FillPolygon	/usr/shlib/X11/lib_dec_ffb_ev5.so
170723	2.80%	65.14%	in_checksum	/vmunix
161326	2.65%	67.78%	miInsertEdgeInET	/usr/shlib/X11/libmi.so
133768	2.19%	69.98%	miX1Y1X2Y2InRegion	/usr/shlib/X11/libmi.so

How does DCPI work?

- Performance Counter Hardware
 - aggregate event counters for in-order CPUs
 - profile-me counters and flags for OOO CPUs
- Device Driver: controls counters and services counter hardware interrupts
- Loader, kloadsrv & exec-path hooks: map samples to proper image
- User-level daemon: consolidates, saves data
- Tools (setup, control, analysis, presentation)

DCPI Components



In-Order Machine profiling

- Count events (e.g. cycles, cache misses)
 - Program counter values and hardware performance registers are periodically sampled.
- Interrupt when counter overflows
- Driver records restart PC -- OK for cycles
- Samples proportional to total time at head of issue queue (could be executing or stalled)
- Any stall / unused issue slot implies performance loss

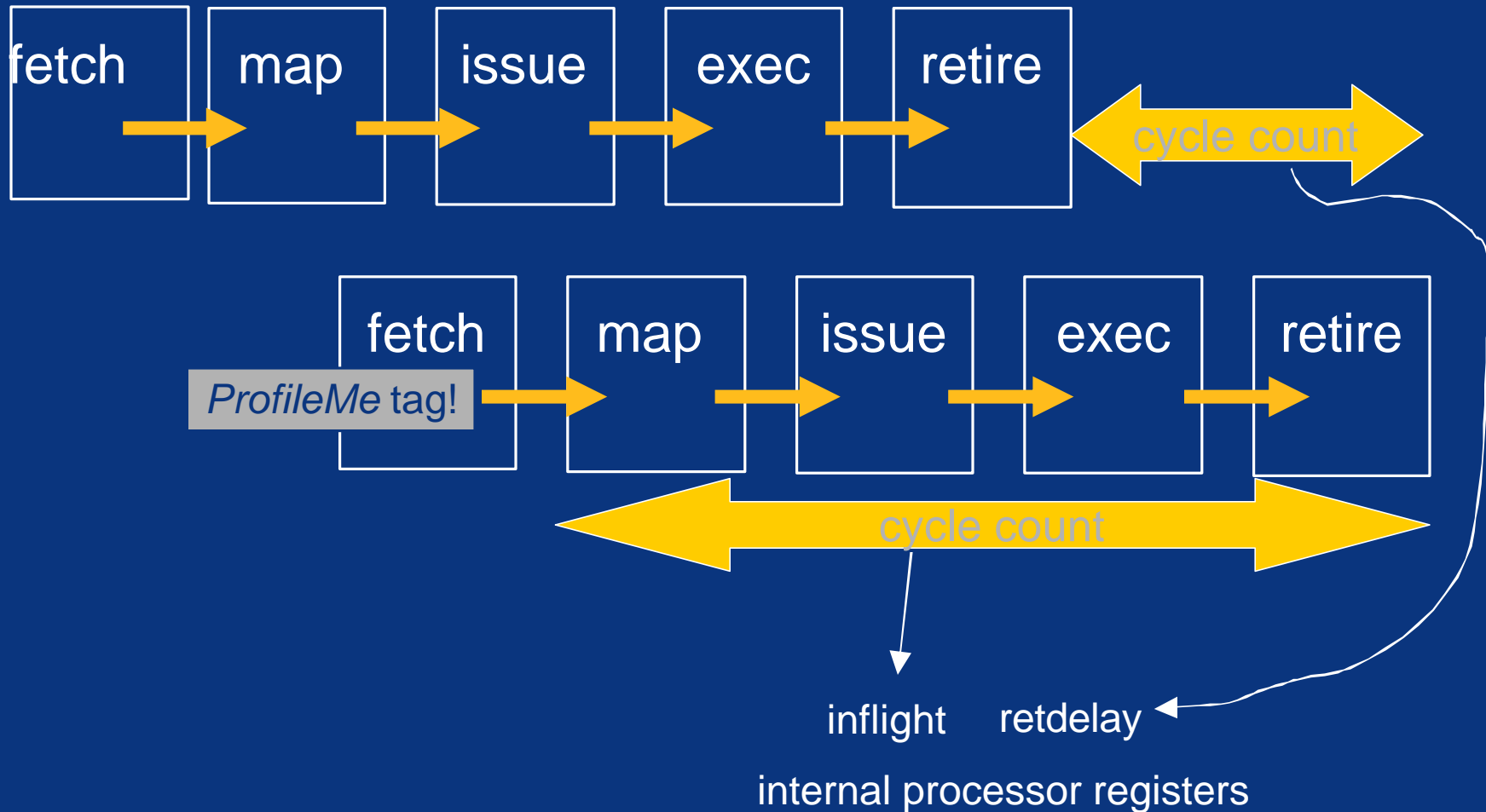
OOO Machine profiling

- Much faster systems, but more sophisticated
- Designed to hide instruction stall latency by exploiting instruction level parallelism
- Instruction stalls not necessarily bad
- Many instructions simultaneously in-flight
- Variable skew prevents precise PC attribution
- Impractical to code machine model
- Instruction level profiling still very important

Profile-me technology

- Sample instructions, don't count events
- Uniformly pick instructions and trace their execution as they flow through pipeline
- PC known from start - no attribution problem
- Collect detailed and exact info for instruction
- Saved in special processor registers
 - (e.g. PC, retired?, retire delay, cache miss?, trap?, trap type)
- *You need at least a minute of execution time to get statistically valid results, five minutes even better*
 - *Run the program 20 times in a row if you have to.*

ProfileMe on EV67: "Counters"



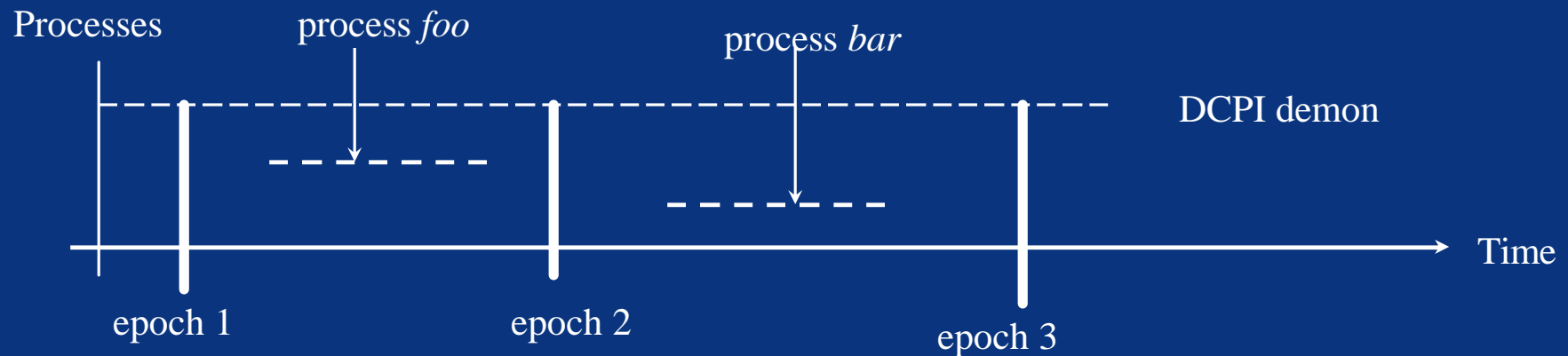
DCPI

DCPI monitors the hardware counters, but lets you look at several statistics at once.

Root starts up the DCPI demon, which samples the hardware counters.

Each user can

- cause the demon to dump all of its statistics to an 'epoch' directory
- examine the statistics of whichever executable is of interest



Performance analysis steps

1. Collect data.
2. Break down the data by image to find the most active program components.
3. Break down the most active program components by procedure to find the most active procedures.
4. Display an instruction level analysis of the most active procedures.
5. Interpret the analysis, find potential issues, make a change and and repeat these steps.

1. Starting Data Collection

Create directory for profile data: db

```
> mkdir ~/db
```

Start profiling daemon with options

```
> dcpid -slot pm -slot cycles ~/db
```

Multiplex between

- ProfileMe data
(instruction-level data)
- Aggregate cycle counts
(no lost cycles)

Running the Workload

Optionally, mark off an epoch before and after you execute the program:

```
> dcpictl epoch
```

```
> hydro
```



```
> hydro
```

```
> dcpictl epoch
```

Ending/Flushing Data Collection

Dcpid buffers profiles in memory.

When workload exits, either

Flush profiles to disk

> **dcpiflush**

Flush profiles to disk and turn off data collection

> **dcpiquit**

2. Profile data by image

dcpiprof program breaks out event profile data

- by binary program image
- by procedure within an image

```
% dcpiprof - event retires:count -t cycles -i
```

One can look for other performance problems by routines:

```
% dcpiprof -event bmiss -i
```

```
% dcpiprof -event replaytrap -i
```

```
% dcpiprof -event mispredict -i
```

```
% dcpiprof -event cycles -i
```

3. Profile data by procedure

`% dcpiprof -pm retired:count -t cycles hydro -db db`

Output:

```
Column                Total  Period (for events)
-----                -
cycles                184782 126976
retired:count         164297 126976
```

the ratio of retired:count / cycles is 164297/184782 < 1

The numbers given below are the number of samples for each listed event type or, for the ratio of two event types, the ratio of the number of samples for the two event types.

```
=====
                                retired
cycles      %      cum%      :count      % procedure      image
61626      33.35%    33.35%    52228      31.79% vsetuv_          hydro
46303      25.06%    58.41%    28118      17.11% vqterm_          hydro
23884      12.93%    71.33%    33533      20.41% srchdf_          hydro
 9911       5.36%    76.70%    10577       6.44% vstden_          hydro
 9650       5.22%    81.92%     6651       4.05% update_          hydro
```

4. Profile data by instructions

You can see line by line using

```
% dcpilist -pm retired:count -t cycles vsetuv_ hydro
```

Output:

retired

:count cycles freq

....

```
74 16 13.6cy+? DO 150 L=L1,L2
0 0 C
339 262 58.2cy+? D2U2=DEL2(D1U(L),D1V(L))/
285 361 51.6cy+? . MAX(DEL2(D2U(L),D2V(L)),F1EM36)
0 0 C
489 853 86.6cy+? UT(K,L)=UT(K,L)-DTN*D2U(L)*XIHMR(L)*D2U2
135 183 23.6cy+? IF(ABS(UT(K,L))-F1EM8.LT.O)THEN
283 673 51.6cy+? UT(K,L) = 0.0
0 0 ENDIF
0 0 C
```

5. Analysis

It helps know potential problems that can cause performance degradation

The best ways to use DCPI is to do hypothesis testing

How well am I doing?

EV67 has

- 4 integer units: retire 4 instruction/cycle
- 2 floating point units: retire 2 instruction/cycle

```
% dcpiprof -i -pm retired:count -t cycles hydro -db db
```

Output:

Column	Total	Period (for events)
-----	-----	-----
cycles	184782	126976
retired:count	164297	126976

the ratio of retired instruction to cycles is $164297/184782 < 1$

There is much room for improvement.

(Can get the same data procedure by procedure)

dcpiprof Default Output

dcpiprof: no images specified. Printing totals for all images.

Image id 0x38ed4a5e00197f9c: no name found for this image.

Image id 0x39a5c561002c46bb: no name found for this image.

Image id 0x38ed48f30078775d: no name found for this image.

Column	Total	Period (for events)
l:retdelay	61770288	190464
l:retired:count	170074726	190464
l:retired:count	1705427	190464
l:notrap:count	70796	190464
l:miss	39191	12288
l:cycles	63506442	190464

The numbers given below are the number of samples for each listed event type or, for the ratio of two event types, the ratio of the number of samples for the two event types.

```
=====
```

valid	retired			!retired			!notrap						
retdelay	%	cum%	:count	%	:count	%	:count	%	bmiss	%	cycles	%	image
3350343	86.37%	86.37%	151842271	89.28%	79925	4.69%	3046	4.30%	16735	42.70%	54500345	85.82%	/vmunix
2024263	3.28%	89.65%	4981351	2.93%	116485	6.83%	8268	11.68%	531	1.35%	2208129	3.48%	
usr/shlib/libelan3.so													
534460	0.87%	90.51%	1040349	0.61%	107026	6.28%	3514	4.96%	931	2.38%	627642	0.99%	
usr/shlib/libelan.so													
285237	0.46%	90.97%	583774	0.34%	55078	3.23%	2189	3.09%	1545	3.94%	299768	0.47%	./fftw


Interpreting The Output

Column	Total	Period (for events)
-----	-----	-----
valid:retdelay	61770288	190464
retired:count	170074726	190464
!retired:count	1705427	190464
!notrap:count	70796	190464
bmiss	39191	12288
cycles	63506442	190464

- Provides information on six different types of events
- Header lists reported events, total number of samples recorded per event type, and sampling period for each event
- For example: 39191 samples of type bmiss were recorded with each sample accounting for 12,288 bmisses on average

Interpreting The Output (cont.)

valid			retired		!retired			
:retdelay	%	cum%	:count	%	:count	%	...	image
53350343	86.37%	86.37%	151842271	89.28%	79925	4.69%	...	/vmunix
2024263	3.28%	89.65%	4981351	2.93%	116485	6.83%	...	/usr/shlib/libelan3.so
534460	0.87%	90.51%	1040349	0.61%	107026	6.28%	...	/usr/shlib/libelan.so
285237	0.46%	90.97%	583774	0.34%	55078	3.23%/fftw



- First three columns are information on event used to sort records
 - first column lists event samples within this image/procedure
 - second column lists the percentage from total samples of dcpiprof's output (i.e. 0.46% fell within ./fftw)
 - third column gives cumulative percentage
- Next columns are secondary events with two columns each
 - 1st number of samples; 2nd percentage of total
- Last is image name

Combining metrics – Multiple Columns

Combine several metrics in one profile

The "+" operator creates additional columns

```
dcpiprof -pm retired+replay -i
```

retired			replays		
:count	%	cum%	:count	%	image
40044	91.71%	91.71%	0	0.00%	sample_v2

Also used for procedures

```
dcpiprof -pm retired+replay sample_v2
```

Combining metrics – Events & Traps

Combine an event and a trap bit setting

- ^ – logical AND
- ! – logical NOT (use \! or substitute "/" for "!")

```
dcpiprof -pm retired^notrap -i
```

- names samples where the instruction retired and didn't cause a trap

```
dcpiprof -pm taken^\!mispredict -i (or /mispredict)
```

- names all samples where a branch was taken and did not mispredict

What went wrong?

Branch mispredictions

- instructions speculatively fetched and executed must be discarded
- processor must roll back to architectural state at time of the branch
- instructions must be fetched from correct target address
- Note the :: operator computes the ratio

```
dcpiprof -pm cbrmispredict::retired heat.single
```

```
dcpilist -pm cbrmispredict::retired mcgds_ heat.single
```

Trap and Abort Rates

```
> dcpiprof $labels $db -i -sp trap::ret -pm /ret::ret *.exe
```

```
Column                               Total   Period (for events)
-----                               -
!notrap:count::retired:count         0.010919   NA
!retired:count::retired:count       0.283479   NA
=====
```

```
!notrap !retired
:count  :count
::      ::
retired  retired
:count  :count image
0.0227  0.5393 javacbase.exe
0.0160  0.5200 dbbase.exe
0.0144  0.2687 jackbase.exe
0.0138  0.2775 mtrtbase.exe
0.0134  0.3188 jessbase.exe
0.0088  0.2368 compressbase.exe
0.0041  0.1132 mpegaudiobase.exe
```

1 trap for every 44 retires!
1 abort for every 2 retires!

Replay Traps

All younger instructions are killed

Instruction is re-fetched, re-mapped

- A **load-store** replay trap occurs when a load, which consumes the value from a store to the same address, has issued before the store. The store is allowed to complete before restarting execution from the load.
- A **load-load** replay trap occurs when two loads from the same address issue out of program order. The loads are forced to complete in program order.
- A **load-miss load** replay occurs when a load misses and it is followed by another load from the same address. The first load is allowed to complete before executing the second load.
- A **wrong size** replay trap is performed when a store is followed by a load that reads the same data and the load data type is larger than the store. The store is allowed to complete before the load is issued.
- A **cache line contention** replay trap occurs when loads and stores that are in progress at the same time and that map to the same cache line (32Kb spacing/stride.) The processor replays instruction that triggered the replay trap until the earlier memory operations have completed. (Trolls)

same dcache index (32k apart)

same bcache index (1M apart)

Replay traps

Need to know how often replay traps occurs

- Trap frequency (X traps/ Y retires)

```
dcpi prof -pm \!notrap:count::retired:count heat.single
dcpilist -pm \!notrap:count::retired:count \
    global_r8_sum_ heat.single
```

Number of instructions aborted

- X aborts / Y retires

```
dcpi prof -pm \!retired:count::retired:count heat.single
dcpilist -pm \!retired:count::retired:count \
    faceput_ heat.single
```

Cache misses

Use the library that show load/store latency, one can infer about cache misses

Start dcpid as

```
dcpid -vtrace '/lib/dcpi/vp-ldlatency.so' db
```

The listing gives histogram of load latencies. Long latencies imply cache misses

```
dcpilist -both -vtrace '/lib/dcpi/vp-ldlatency.so' -p someproc hydro
```

dcpid

dcpid {-event [event_type]} -gc -byid a.out /local/dcpid/db

-gc : deletes all previous profiles except last 3 epochs

-byid : creates profile files per each process executing a.out

-slot : indicates events to be profiled

(use one pair '-slot [event_type]' for each event)

event_type: possible events

cycles = processor cycles (c0 or c1)

retires = retired instructions (c0)

replaytrap = mbox replay traps (c1)

itbmiss = retired itb misses (c1)

dtbmiss = retired dtb single misses (c1)

dtbdblmiss = retired dtb double misses (c1)

retcondbr = retired conditional branches (c1)

retunalign = retired unaligned traps (c1)

dcpiprof & dcpilist

dcpiprof -epoch latest-1 [options] a.out > log.profile

dcpiprof -profiles {[pid_prof_file] } -- [options] a.out > log.profile

options :

- profiles: indicates profile file (good for multiple pids)
- st [event_type] (event used as sort key)
- allevents (lists all events that were profiled)
- events [event_type+event_type...] (particular events)
- events all-[event_type] (lists all except the indicated)
- keep [p] (lists only top p% (in cycles) of the events)

dcpilist [options] [procedure_name] a.out > log.list

dcpilist [options] -p {[procedure name]} -- a.out > log.list

dcpilist -profiles {[pid_prof_file]} -- [options] [procedure] a.out > log.list

options:

- profiles: indicates profile file (good for multiple pids)
- f [sourcefile]: indicates which source file to use
- source : prints only the source code
- asm : prints disassembly code
- both : lists source and assembly code
- order : preserves better the execution order
- lines : associates to line number in source file

Issues for DCPI Profiling on SC

There are problems running DCPI on multiple SC nodes

Errors opening several DCPI databases on NFS

Allocate command often disabled by SC administrators

All jobs get killed when prun finishes

Workaround: put profile locally on each machine

dcpi_sc Script

```
#!/bin/csh
#dcpi_sc
prun -N 1 -r csh -c "/usr/bin/dcpiscan . > ./dcpi.map"
prun -N $1 -r /hpc2/niraj/bin/start_dcpid
prun -N $1 -r -t ps aux | grep dcpid
```

Must be run as root

Maps the images

Runs script start_dcpid to start daemon

Verifies dcpid started

start_dcpi Script

```
#!/bin/csh
# start_dcpi
set name=`hostname`
rm -rf /tmp/DB.*
mkdir /tmp/DB.$name
chmod -R 777 /tmp/DB.$name
setenv DCPIDB /tmp/DB.$name
dcpid -no_default_scanmap -slot cycles -slot bmiss -slot \ pm -m
./dcpimap $DCPIDB >& dcpimsg.$name
```

Uses node hostname for database name

Starts dcpi daemon

dcpi_clean Script

```
#!/bin/csh
#postprocess
set name=`hostname`
setenv DCPIDB /tmp/DB.$name
dcpiquit >>& dcpimsg.$name
dcpiprof >& dcpilog.all.$name
dcpiprof -s cycles /vmunix | head -50 >& dcpilog.vm.$name
```

After the run, stop the daemon and flush data

Send default profile values to file

Send specific profile values to another file

In example, cycles in operating system vmunix

Phoenix Graphical User Interface for DCPI

What is Phoenix?

Phoenix is a tool which is intended to make DCPI easier to use

- Lowers amount of knowledge needed to effectively use DCPI, and
- Allows power users to become more productive

X-window display

Does not replace DCPI -- works with it

Executes same underlying DCPI user commands

Displays DCPI commands for cut-and-paste

Works on Tru64 and Linux

Phoenix Main Window

Daemon control

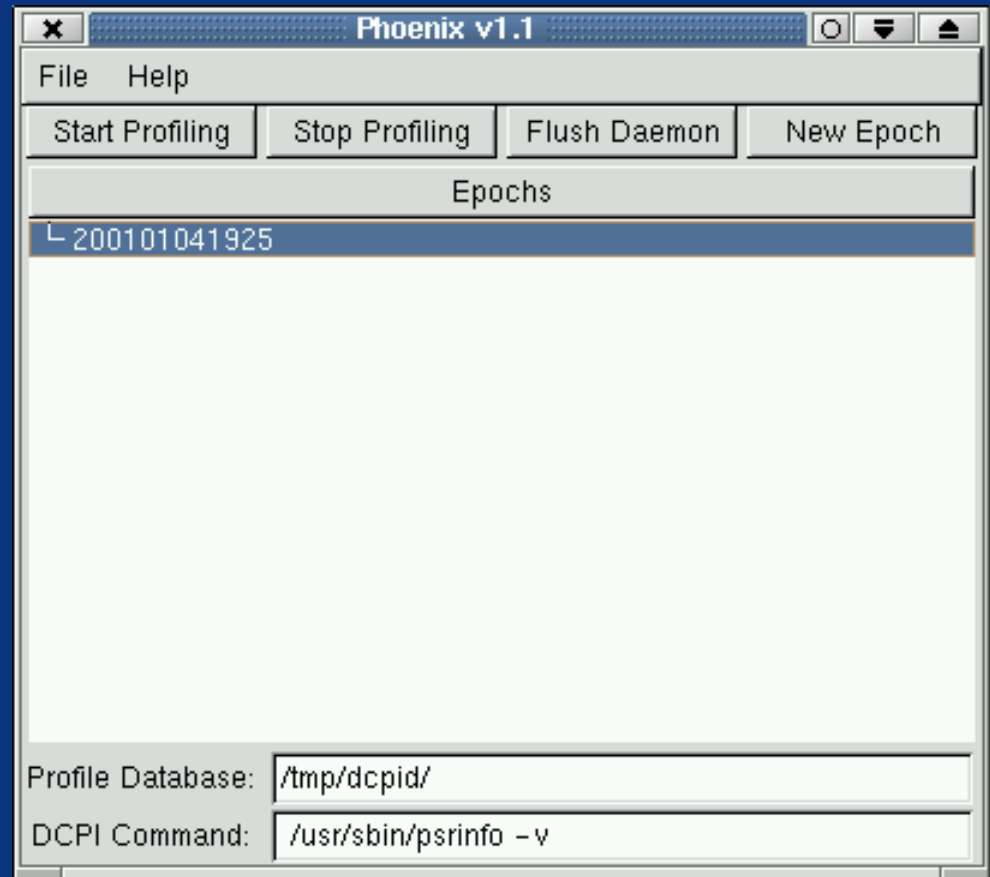
- Start / stop profiling
- Flush profile to disk
- Create new epoch

Double click epoch name to bring up Navigation window

Right click epoch to rename or delete

Database directory and DCPI command displayed

- Directory can be changed



DCPI Start Window

Pull downs control options

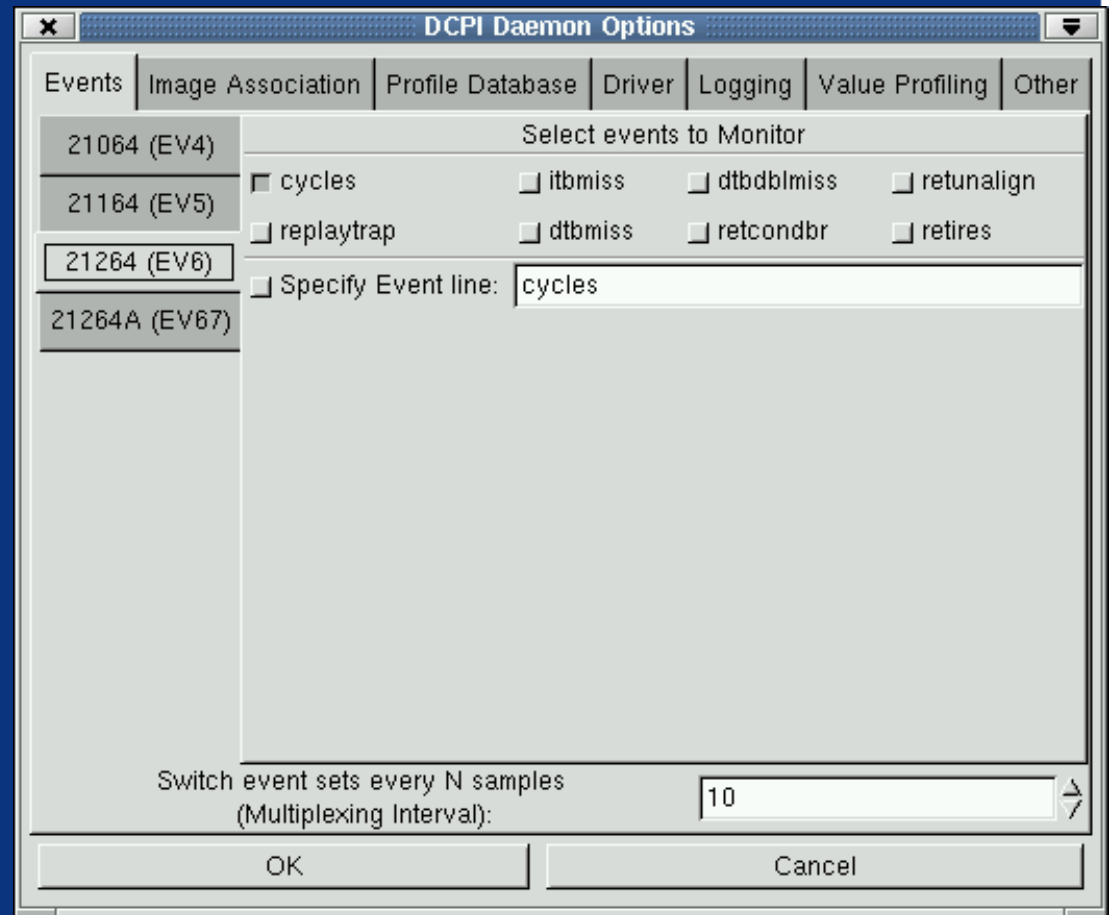
Select right architecture

- Phoenix may select it
- Must be correct

Select events to monitor

Or enter events in window

Enter interval for multiplexing events



Epoch Navigator Window

200101041925 - Epoch Name												
valid:retdelay	%	cum%	cycles	%	retired:count	%	lretired:count	%	!notrap:count	%	image	
472957632	48.82	48.82	495603904	48.12	930587277	48.96	9011104	7.21	646392	8.41	/vmunix	
410626260	86.82	86.82	417677355	84.28	840926597	90.37	55213	0.61	3639	0.56	idle_thread	
26946805	5.70	92.52	28419670	5.73	69871052	7.51	1098527	12.19	40074	6.20	alpha_delay	
5131051	1.08	93.60	2562788	0.52	715208	0.08	237908	2.64	36214	5.60	simple_lock	
2732087	0.58	94.18	576900	0.12	255883	0.03	65206	0.72	1778	0.28	simple_unlock	
1736200	0.37	94.55	2500039	0.50	172827	0.02	689140	7.65	12858	1.99	find_bfap	
1691630	0.36	94.91	1910215	0.39	967710	0.10	595066	6.60	20015	3.10	nofault_bcopy	
1527450	0.32	95.23	2201791	0.44	2782716	0.30	203341	2.26	5591	0.86	event_timeout	
1509628	0.32	95.55	120331	0.02	49309	0.01	13549	0.15	349	0.05	read_io_port	
999669	0.21	95.76	17060	0.00	4651	0.00	203	0.00	190	0.03	tlaser_pci_eoi	
902480	0.19	95.95	228898	0.05	69530	0.01	23330	0.26	1264	0.20	hw_sg_load	
831903	0.18	96.13	1144311	0.23	888800	0.10	237438	2.63	59197	9.16	simple_lock_retry	
37019	4.45		121480	10.62	59728	6.72	6	0.00	0	0.00	0xffffc00004841b0 ldq t0, 16(a0)	
0	0.00		0	0.00	58799	6.62	2	0.00	0	0.00	0xffffc00004841b4 bis a0, a0, t0	
22948	2.76		0	0.00	59091	6.65	6	0.00	0	0.00	0xffffc00004841b8 ldq a0, 8(a0)	
0	0.00		0	0.00	59666	6.71	3	0.00	0	0.00	0xffffc00004841bc bis a1, a1, t0	
66	0.01		59586	5.21	58702	6.60	0	0.00	0	0.00	0xffffc00004841c0 lda t3, 16384(t0)	
19	0.00		0	0.00	59476	6.69	1	0.00	0	0.00	0xffffc00004841c4 bic sp, t3, t1	
0	0.00		0	0.00	58923	6.63	1	0.00	0	0.00	0xffffc00004841c8 ldah t4, 0(gp)	
96105	11.55		0	0.00	59591	6.70	1	0.00	0	0.00	0xffffc00004841cc ldq t4, -231(t0)	
165877	19.94		59425	5.19	58841	6.62	1	0.00	4	0.01	0xffffc00004841d0 bne t4, 0xffffc00004841d4, :ldl_1	
93026	11.18		59515	5.20	59146	6.65	1	0.00	0	0.00	0xffffc00004841d4 :ldl_1 a1, 0(a0)	
0	0.00		60106	5.25	59258	6.67	1	0.00	0	0.00	0xffffc00004841d8 bis t0, t0, v0	
0	0.00		0	0.00	59186	6.66	1	0.00	59186	99.98	0xffffc00004841dc bic a1, 0xffffc00004841e0, a1	

DCPI Command: `d.count -pm /notrap:count -epoch "200101041925" -db "/tmp/dcpid/" "simple_lock_retry" /vmunix 2>&1`

Navigator Window Fields

File pull down – generate report / close window

Event pull down – ProfileMe menus and displays, e.g.

- All ProfileMe traps
- Data traps
- Branch prediction events
- Custom ProfileMe – brings up new window (see below)
- Show program source code

Percent pull down – control display of percent values

Sample data

- Double click a line to drill down
- Image -> function -> source lines -> assembler lines

Command line – last DCPI command executed

ProfileMe Filter Window

Column assignment

- Pick event set for each column
- Ratio if event above and below “/”

Event sets

- Two events (Event1 and Event2) can be specified
- Relationship of the two events chosen

Column 1	Column 2	Column 3	Column 4	Column 5
Event Set1	Event Set2	Event Set3	Event Set4	None
/	/	/	/	/
None	None	None	None	None
Event Set 1: <input type="checkbox"/> NOT	Event1: valid	AND <input type="checkbox"/> NOT	Event2: None	counter: retdelay
Event Set 2: <input type="checkbox"/> NOT	Event1: retired	AND <input type="checkbox"/> NOT	Event2: None	counter: count
Event Set 3: <input checked="" type="checkbox"/> NOT	Event1: retired	AND <input type="checkbox"/> NOT	Event2: None	counter: count
Event Set 4: <input checked="" type="checkbox"/> NOT	Event1: notrap	AND <input type="checkbox"/> NOT	Event2: None	counter: count

References

External website

<http://www.tru64unix.compaq.com/dcpi/>



i n v e n t