Session III:
Level II Tools: DynaProf
11:00AM – 12:00 PM

Philip J. Mucci
mucci@cs.utk.edu
Innovative Computing Laboratory
University of Tennessee

Lawrence Berkeley National Laboratory
pjmucci@lbl.gov

Outline

DynaProf Introduction

- Goal and Overview
- Architecture

Obtaining and Installing DynaProf

Using DynaProf

- Instrument running executable
- Collect and browse performance data

DynaProf Current Status

Goals

Make collection of hardware performance data quick and straightforward.

Work with production executables as delivered. No recompilation required.

Provide highly accurate hardware performance data with a minimum of interference and overhead.

Provide full platform and language independence.

Methodology

Avoid parsing and recompilation of the source code to insert instrumentation.

Avoid interference of instrumentation with various compiler optimizations.

Allowing multiple insert/remove instrumentation cycles.

Allow the use of the same tool with a number of different performance probes.

PERC

DynaProf SC2003 Release

Supported Platforms with DynInst 3.0 or 4.0

- Linux 2.4+
- Solaris 2.8+
- IRIX 6.x

Supported Platforms with DPCL (fully interactive MPI support)

- AIX 5

URL: http://www.cs.utk.edu/~mucci/dynaprof

Mailing List: perfapi-devel@ptools.org

Summary

DynaProf is a portable tool to gather hardware performance data at run time for an unmodified application.

Instrumentation is done through the dynamic insertion of function calls to specially developed performance probes.

DynaProf provides a simple and intuitive command line interface.

DynaProf Installation

Download appropriate DynaProf binary distribution from web site and follow the instructions.

http://www.cs.utk.edu/~mucci/dynaprof

Requirements:

PAPI 2.x

GNU Readline

On AIX 4.3/5

DPCL (See /usr/lpp/ppe.dpcl)

PMAPI (See /usr/pmapi/*, /usr/lib/libpmapi.a)

On Linux 2.x

DynInst 3.0, 4.0 or later

May require a new binutils, libdwarf, libelf

History of Dynamic Instrumentation

Ideas proposed by James Larus with EEL: An Executable Editor Library at U. Wisconsin

http://www.cs.wisc.edu/~larus/eel.html

Technology developed by Dr. Bart Miller at U. Wisconsin and Dr. Jeff Hollingsworth at U. Maryland.

http://www.dyninst.org

IBM developed a distributed DynInst called DPCL. It is integrated with AIX's parallel runtime system.

http://oss.software.ibm.com/dpcl

PERC

Performance Probes

Papiprobe

Measure PAPI preset and native events.

Papiclock

Measure PAPI real-time and virtual-time cycles.

Wallclock

Measure real-time units.

Perfometerprobe

Enable real-time visualization of PAPI event rates with Perfometer.

Vmonprobe

Collect statistical performance data for vprof.

Using Performance Probes

DynaProf comes with special shared libraries that are the probes.

The functions in these files have a defined interface and calling convention.

Probes are handled by the 'use' command.

Performance Probes

Three probes provide the ability to instrument specific regions of code.

Papiprobe

Papiclock

Wallclock

These probes generate the following data for each instrumented function:

Inclusive: $T_{function} = T_{self} + T_{children}$

Exclusive: $T_{function} = T_{self}$

1-Level Call Tree: T_{child} = ***Inclusive*** $T_{function}$

Performance Probe Data

The papiprobe, papiclock and wallclock probes produce data in an identical format.

These three probes always measure the entire executable "TOTAL" in addition to any additional instrumentation points the user has specified.

All use a processing script to display the data in a human readable format.

papiproberpt <file>

papiclockrpt <file>

wallclockrpt <file>

Papiprobe

By default, it measures PAPI_FP_INS or PAPI_TOT_INS if the former is not available.

Takes a comma separated list of options or PAPI events, either preset or native.

Passing 'help' as option prints out list of available PAPI presets.

Passing 'mpx' or “multiplex” as an option enables the use of counter multiplexing if needed.

Making Sense of Papiprobe Data

Sometimes the data might not make sense.

We need to understand the EXACT semantics of the events.

There is a command that will list all the available PAPI events and their native mappings.

Type " papi_avail -a<cr>"

Note the information at the end of each line between the parenthesis. This can be cross-referenced with that in /usr/pmapi/lib/POWER3.evs, /usr/pmapi/lib/POWER3-II.evs and the RS6000 Architecture Manual. Isn't that fun?

DPCL Issues

Occasionally DPCL and/or application processes will be left stranded. They may continue to run or just block. These issues have not been resolved.

When things get funky:

Type “ dpcl-cleanup<cr>”

This will go out to all 16 nodes on hockney and kill all of your processes, including the DPCL daemon processes and any applications.

For Load Leveler jobs you should continuously monitor your application's progress with:

Type “ llps <username><cr>”

DynaProf Exercises 1, 2 & 3

We will use DynaProf to evaluate different versions of SWIM, the shallow water benchmark code.

1) Discover delivered MFLOP/S and IPC of an entire serial run.
2) Evaluate memory subsystem efficiency (L1HR,L2HR,stall %) of the core compute solvers of a serial run.
3) Measure many events on an MPI executable in batch mode through Load Levelor.

Exercise Preparation

Load the tutorial module:

Type “ cd ~<cr>”

Type “ module load perc.sc03<cr>”

For csh: Type “ source ~/DynaProf/setup.csh<cr>”

For sh: Type “ . ~/DynaProf/setup.sh<cr>”

Type “ echo '+ $USER' > ~/.rhosts'”

Type “ chmod 600 ~/.rhosts<cr>”

All exercises start from the DynaProf directory!

Build the swim and swim_ompi executables.

Type “ cd swim; make<cr>”

Exercise 1: Global MFLOP/S & IPC

Type “dynaprof<cr>”

Type “load swim<cr>”

Type “use papiprobe PAPI_TOT_CYC, PAPI_FP_INS, PAPI_TOT_INS<cr>”

Type “run<cr>”

Type “quit<cr>”

Note name of the output file at beginning of run.

Type “papiproberpt <output_file> | more<cr>”

Exercise 1 cont.

Compute MFLOP/S & IPC:

CPU Seconds = PAPI_TOT_CYC/(Mhz*1.0e6)

TMFLOP = PAPI_FP_INS/(1.0e6)

MFLOP/S = TMFLOP/Seconds

IPC = PAPI_TOT_INS/PAPI_TOT_CYC

Hockney: A 200Mhz Power 3, 2 floating point Instructions/Cycle for 400 MFLIP/S, 800 MFLOP/S and 8 total Instructions/Cycle

PERC

Exercise 1 Answers

CPU Seconds = 3.82e9/(200*1.0e6) = 19.11

TMFLOP = 1.53e9/(1.0e6) = 1530

MFLOP/S = 1530/19.11 = 80

IPC = 3.04e9/3.83e9 = 0.8

Datafile is swim.data.ex1

Report is swim.report.ex1

Other Things to Try:

Listing the available PAPI events.

Type " use papiprobe help" to DynaProf

Use multiplexing with lots of PAPI events.

Type " use papiprobe mpx, <event>, ..."

Use only with large granularity measurements!

Attaching to a process instead of loading:

Type " attach <exe> <pid>"

PERC

Exercise 2: Routine-Level Memory Effects

Measure solver routines to get % memory load stall cycles and % load miss rates.

Type “ dynaprof<cr>”

Type “ load swim<cr>”

Type “ list<cr>”

Type “ list module swim.F<cr>”

Type “ list functions swim.F calc*<cr>”

Type “ list children swim.F inital<cr>”

Type “ list children swim.F shalow<cr>”

DynaProf Command Line Editing

Provides robust command line editing

- Arrow Keys and Emacs Bindings:
  - Delete char under cursor
  - C-a Beginning of Line
  - C-e End of line
  - C-<spc> Set mark
  - C-w Cut to mark
  - C-y Yank cut text
- <TAB> triggers filename completion

Exercise 2 cont.

Type “ use papiprobe PAPI_TOT_CYC, PAPI_MEM_RCY, PAPI_L2_LDM, PAPI_LD_INS, PAPI_L1_LDM<cr>”

Type “ instr function swim.F calc*<cr>”

Type “ run<cr>”

Type “ quit<cr>

Note name of the output file at beginning of run.

Type “ papiproberpt <output_file> | more<cr>”

Exercise 2 cont.

Compute % Memory Load Stall Cycles:

% Stall = 100.0 * (PAPI_MEM_RCY/ PAPI_TOT_CYC)

Compute L1 Miss Rate:

% L1 Miss = 100.0 * (PAPI_L1_LDM/PAPI_LD_INS)

Compute L2 Miss Rate:

% L2 Miss = 100.0 * (PAPI_L2_LDM/PAPI_LD_INS)

Exercise 2 Answers

Compute % Memory Load Stall Cycles:

Calc3: 100.0*(8.96e8/1.34e9) = 67.01 %

Calc2: 100.0*(7.11e8/1.26e9) = 56.43 %

Calc1: 100.0*(3.38e8/1.05e9) = 32.19 %

Compute L1 Miss Rate:

Calc 3: 100.0 * (1.75e7/2.79e8) = 6.27 %

Calc 2: 100.0 * (1.4e7/4.25e8) = 3.29%

Calc 1: 100.0 * (6.07e6/2.52e8) = 2.40%

Compute L2 Miss Rate:

Calc 3: 100.0 * (844/2.79e8) = < 1%

Calc 2: 100.0 * (2/4.25e8) = < 1%

Calc 1: 100.0 * (0/2.52e8) = < 1%

Exercise 2: Exclusive Profile of Cycles

Name	Percent	Total	Sub. Calls
TOTAL	100	3.836e+09	1
calc3	34.92	1.34e+09	118
calc2	32.89	1.262e+09	120
calc1	27.44	1.053e+09	120
unknown	4.471	1.715e+08	1
calc3z	0.2798	1.073e+07	1

Exercise 2: 1-Level Incl. Call Profile of Cycles

Name	Percent	Total	Sub. Calls
TOTAL	100	3.836e+09	1
calc1	100	1.053e+09	120
calc2	100	1.262e+09	120
-fsav	0.02512	3.171e+05	120
calc3z	100	1.073e+07	1
calc3	100	1.34e+09	118

DynaProf and Threads

For threaded code, just specify the the same probe!

DynaProf detects a threaded executable and loads a special version of the probe library.

The probe detects thread creation and termination.

All threads share the instrumentation.

Output goes to <exe>.<probe>.<pid>.<tid>

DynaProf and MPI

On AIX with DPCL, DynaProf talks directly to the parallel run-time system. (POE)

poeattach <exe> <pid_of_poe>

poeload <exe> <poe args>

With DynInst, DynaProf must be run in batch mode as part of the line to mpirun. DynaProf provides a special load that waits until MPI_Init() returns before continuing.

mpiload <exe> <args>

DynaProf Batch Mode

DynaProf can run from a script via command line arguments:

-c <FILE>	Specifies the name of a script
-b	Exits after processing the script
-q	Suppress printing any output

You can see all DynaProf's arguments by using the -h flag.

All arguments have long versions.

Exercise 3: Instrument an MPI Application

Edit the DynaProf script.

Type " vi swim_mpi.ex3.dp<cr>"

Edit the Load Leveler script.

Type " vi swim_mpi.ex3.ll<cr>"

Type " llsubmit swim_mpi.ex3.ll<cr>"

Look in "swim_mpi.ex3.out" for name of probe output files

Type "papiproberpt <output_file> | more<cr>"

DynaProf Development

Changes to the DynInst version:

- ***Port to DynInst on AIX5 and Linux/IA64***
- ***Interactive instrumentation of MPI codes with Client/Server framework***
- ***New instrumentation support: Object/Loop/Basic Block/Arbitrary***
- ***Breakpoints***
- ***Dump instrumentation data upon demand.***
- ***Multiple insert/remove cycles within the same run.***
- ***Handle programs that load extensions at run-time. (i.e. Mozilla)***

Probes

- ***Additional thread model support for IRIX, Tru64 and Solaris.***
- ***Thread support in vprof, perfometer probe***
- ***Integration with ParaProf from TAU for visualization of probe data.***

PERC

Analysis of POP

Serial, 4 and 16 processor runs.

Both optimized and debug versions.

PAPI data

- Entire run
- Routine-specific

Getting the data was very difficult:

- **DPCL interferes with shared memory to adapter so we need to use IP for communication. Ugh! This is not a problem on machines with more memory on a node, like seaborg.**

POP Routine Breakdown

From previous production runs we know:

Function:	Calls	Excl.	Incl.	Descr.
tracer_update	800	39	55	update tracer fields at level k
state	6520	35	35	calc. density of water at level k
clinic	800	34	37	calculate forcing terms on r.h.s. of baroclinic momentum
baroclinic_driver	20	31	136	explicit time integration of baroclinic velocities
pcg	20	18	18	conjugate-gradient solver w/precond.
Impvmixt	39	9	9	implicit vertical mixing of tracers

POP Routine Breakdown by File

25.5%	17.2%	20.6%	state_mod.f
18.0%	15.7%	16.8%	hmix_gm.f
12.3%	13.8%	13.7%	hmix_aniso.f
9.7%	10.4%	10.2%	vmix_kpp.f
7.8%	8.9%	8.3%	stencils.f
3.5%	7.5%	6.3%	vertical_mix.f
4.7%	6.2%	5.6%	advection.f
3.6%	5.6%	4.8%	baroclinic.f
4.4%	2.5%	3.2%	../../../../../../../src/bos/usr/ccs/lib/libm/POWER/cosF.c
2.4%	3.6%	3.1%	solvers.f
4.1%	2.3%	3.0%	../../../../../../../src/bos/usr/ccs/lib/libm/POWER/sinF.c
0.9%	1.4%	1.1%	tavg.f
0.6%	1.2%	0.9%	pressure_grad.f
0.8%	0.7%	0.8%	global_reductions.f

POP Routine Breakdown by Function

25.5%	17.2%	20.6%	.__state_mod_MOD_state
18.0%	15.5%	16.8%	.__hmix_gm_MOD_hdifft_gm
12.3%	13.8%	13.7%	.__hmix_aniso_MOD_hdiffu_aniso
8.6%	5.0%	6.2%	<unknown>
2.9%	4.2%	3.7%	.__stencils_MOD_hupw3
1.8%	3.9%	3.2%	.__vertical_mix_MOD_impvmixt
2.4%	3.5%	3.1%	.__solvers_MOD_pcg
3.1%	2.7%	2.8%	.__vmix_kpp_MOD_blmix
2.1%	2.4%	2.3%	.__advection_MOD_advt_upwind3
2.3%	1.9%	2.1%	.__stencils_MOD_ninept_4
1.7%	2.0%	2.0%	.__vmix_kpp_MOD_ri_iwmix
2.0%	2.0%	2.0%	.__vmix_kpp_MOD_wscale
1.4%	1.8%	1.7%	.__advection_MOD_advu
1.2%	1.9%	1.7%	.__baroclinic_MOD_tracer_update

POP Environment

Hardware: 2-way 200 Mhz. Power 3 Nodes

Memory: L1 32K/64K, L2 4MB, 2GB/node

Mpxlf Compiler/Runtime: 8.1.0.0

Flags:

-O3 -qstrict -qarch=auto -bmaxdata:0x80000000

Poe version: 3.2.0.14, css0, shared, ip

DynaProf Reports for POP

Change to one of the executable directories:

1proc-serial, 4proc, 16proc

Browse the data files where they exist.

Type “ ls pop-opt-all<cr>”

Type “ ls pop-opt-routine<cr>”

Type “ ls pop-debug-all<cr>”

Type “ ls pop-debug-routine<cr>”

Exercises 4 & 5: POP Performance

We will use a prebuilt, optimized version of POP for 2 processors, that runs for only 2 time steps.

Type “ cd 2proc-quick<cr>”

Exercises:

1) Measure entire run
2) Measure bottleneck routines

Exercise 4: Overall Performance

Type “vi pop-opt.ex4.dp<cr>”

Type “vi pop-opt.ex4.ll<cr>”

Type “llsubmit pop-opt.ex4.ll<cr>”

Type “ls pop-opt.papiprobe.[0-9]*<cr>”

Type “papiproberpt <file> | more<cr>”

Exercise 4: LoadLeveler Script

```
#@ job_name              = pop-opt-ex4-dynaprof-2proc-quick
#@ account_no          = perc
#@ class                = regular
#@ job_type             = parallel
#@ output               = pop-opt.ex4.out
#@ error                  = pop-opt.ex4.err
#@ wall_clock_limit     = 00:05:00
#@ network.MPI          = css0,not_shared,ip
#@ environment          = COPY_ALL
#@ notification         = never
#@ node                 = 1
#@ tasks_per_node       = 2
#@ shell                  = /usr/bin/csh
#@ queue
#
dynaprof -q -b -c pop-opt.ex4.dp
```

Global POP Performance for 1 CPU Run: x1 Data, 2x2 Procs, 10 Steps

Raw Data	Debug	Optimized
PAPI_LD_INS	1.21E+011	2.104E+10
PAPI_SR_INS	2.02E+010	7.783E+09
PAPI_BR_INS	8.64E+009	5.043E+09
PAPI_FP_INS	2.21E+010	2.251E+10
PAPI_FMA_INS	1.04E+010	1.007E+10
PAPI_FPU_FDIV		2.551E+08
PAPI_FPU_FSQRT		1.317E+08
PAPI_TOT_INS	3.28E+011	6.257E+10
PAPI_TOT_CYC	3.63E+011	6.226E+10
PAPI_L1_LDM	1.03E+009	1.011E+09
PAPI_L1_STM	3.54E+008	3.475E+08
PAPI_L2_LDM	6.94E+008	6.894E+08
PAPI_FPU_IDL	1.66E+011	1.411E+10
PAPI_LSU_IDL	4.06E+010	1.483E+10
PAPI_MEM_RC	1.03E+011	1.368E+10
PAPI_MEM_SC	1.26E+011	2.413E+10
PAPI_STL_CCY	2.01E+011	3.367E+10

Metric	Debug	Optimize
% Ld Ins	36.86	33.63
% Sr Ins	6.17	12.44
% Br Ins	2.63	8.06
% FP Ins	6.75	35.98
% FMA Ins	3.16	16.09
% FP Divide		0.41
% FP SQRT		0.21
MFLIPS	12.19	72.31
% MFLIPS Peal	3.05	18.08
IPC	0.90	1.00
Mem Opts/FLIF	6.38	1.28
% L1 Ld HR	99.15	95.19
% L1 Sr HR	98.25	95.54
% L2 Ld HR	99.43	96.72
% FPU Idle Cyc	45.77	22.66
% LSU Idle Cyc	11.17	23.82
% Ld Stall Cyc	28.28	21.97
% Sr Stall Cyc	34.59	38.76
% No Ins. Cyc	55.25	54.08

POP Global Data Conclusions

L1/L2 Hit Rates are reasonable

But 50% of cycles do not complete an instruction, regardless of optimization settings

And less than half of all floating point instructions are FMA's.

Optimized code is stalled on stores nearly 40% of the time

Optimizer does a good job of eliminating redundant loads/stores.

Exercise 5: Routine Performance

Type “vi pop-opt.ex5.dp<cr>”

Type “vi pop-opt.ex5.ll<cr>”

Type “llsubmit pop-opt.ex5.ll<cr>”

Type “ls pop-opt.papiprobe.[0-9]*<cr>”

Type “papiproberpt <file> | more<cr>”

Exercise 5 cont.

AIX/Fortran 90 name mangling consist of 3 parts:

Name of source file with 2 prepended underscores

MOD

Name of function

Example: vertical_mix.f, subroutine impvmixt

__vertical_mix_MOD_impvmixt

Exercise 5: DynaProf Script

```
poeload pop-opt

use papiprobe PAPI_TOT_CYC, PAPI_STL_CCY, PAPI_STL_ICY,
    PAPI_MEM_RCY, PAPI_MEM_SCY, PAPI_LD_INS,
    PAPI_SR_INS, PAPI_TOT_INS

instr module baroclinic.f

instr module solvers.f

instr module hmix_gm.f

instr module horizontal_mix.f

instr module vertical_mix.f

instr module vmix_kpp.f

instr function state_mod.f __state_mod_MOD_state

run
```


Exercises 4 & 5: Answers

Exercise 4

Type “ more pop-opt.proc*.report.ex4<cr>”

Exercise 5

Type “ more pop-opt.proc*.report.ex5<cr>”

POP Routine Performance for 1 CPU
Run: x1 Data, 8x2 Procs, 10 Steps

Exclusive Profile Timings

PAPI_TOT_CYC	
hdifft_gr	19.51
State	18.06
Pcg	3.41
Impvmix	3.26
Blmix	2.83
ri_iwmix	1.84
Wscale	1.74

PAPI_STL_ICY	
hdifft_gr	23.07
State	16.5
Pcg	2.04
Impvmix	4.23
Blmix	2.79
ri_iwmix	2.07
Wscale	2.1

PAPI_MEM_SCY	
hdifft_gr	28
State	16.4
Pcg	3.3
Impvmix	3.13
Blmix	3.22
ri_iwmix	2.02
Wscale	0.85

PAPI_SR_INS	
hdifft_gr	15.58
State	19.33
Pcg	6.53
Impvmix	2.47
Blmix	1.82
ri_iwmix	2.34
Wscale	1.18

PAPI_STL_CCY	
hdifft_gr	23.04
State	16.81
Pcg	2.42
Impvmix	4.15
Blmix	2.74
ri_iwmix	2.05
Wscale	2.03

PAPI_MEM_RCY	
hdifft_gr	35.39
State	9.27
Pcg	4
Impvmix	3.56
Blmix	4.01
ri_iwmix	0.95
Wscale	0.24

PAPI_LD_INS	
hdifft_gr	16.49
State	19.32
Pcg	4.77
Impvmix	1.8
Blmix	2.14
ri_iwmix	1.44
Wscale	1.19

PAPI_TOT_INS	
hdifft_gr	14.73
State	23.59
Pcg	3.45
Impvmix	2.25
Blmix	3.58
ri_iwmix	1.75
Wscale	1.58

POP Routine Data Conclusions

Stalls are overwhelming in the subroutines hdifft_gm and state.

State, while having twice as many instructions, takes up about the same amount of time.

hdifft_gm calls NO SUBROUTINES or LIBRARY FUNCTIONS. All computation is explicit, thus it is a good candidate for hand-optimization or library replacement.

PERC

References

DynaProf and PAPI

http://www.cs.utk.edu/~mucci/dynaprof

http://icl.cs.utk.edu/projects/papi

DynInst

http://www.dyninst.org

http://www.paradyn.org

DPCL

http://oss.software.ibm.com/dpcl

PERC

References 2

GNU Binutils

http://ftp.gnu.org/gnu/binutils

http://sources.redhat.com/binutils

GNU Readline

http://cnswww.cns.cwru.edu/~chet/readline/rltop.html

http://ftp.gnu.org/gnu/readline

References 3

Libdwarf - DWARF Debugging Library

http://reality.sgi.com/davea

Libelf – ELF Object File Access Library

http://www.stud.uni-hannover.de/~michael/software/english.html

Acknowledgments

This work was supported by DOE SciDAC via PERC

All PERC members contributed in some form or another

We wish especially to thank

Today’ s assistants:

Tushar Mohan

Pat Worley

Ying Zhang

David Skinner from NERSC for helping with accounts

Tushar for setting up the modules and web page

Thank You.