CrayPat – Cray X1 Performance Analysis Tool

Steve Kaufmann

Bill Homer

14 May 2003

Introduction

- Cray X1 introduces challenges in determining performance bottlenecks
- Single- or multiple-PE applications that use shared-memory and distributed-memory parallelism models
- Users need tools that assist in locating opportunities for performance improvements
- Users need to understand how software and hardware resources are utilized

Introduction (cont'd)

- CrayPat is a new tool of multiple components
- Cray PVP and Cray MPP had many tools
- Single “point of entry” for performance analysis
- Steps: instrumenting, executing, reporting
- Supports multiple *experiments*
- Asynchronous (interrupt-based)
- Synchronous (trace-based)
- Stand-alone utilities – **pat_hwpc**

Hardware Performance Counters

- **pat_hwpc** collects HWPC information for an application
- No instrumentation is required
- Similar to Cray PVPs **hpm** utility
- Accepts various HWPC groups
- Results in report with raw counts and derived statistics for the whole application
- Access to over 200 events from three chips
- HWPC summed across threads per process

Instrumented Experiment Types

- Asynchronous (single instrumentation, run-time choice, statistical in nature)
  - **profil**, **mprofil**
  - **samp_pc_time, samp_cs_time, samp_heap_time, samp_ru_time**
  - **samp_pc_ovfl, samp_cs_ovfl, samp_heap_ovfl, samp_ru_ovfl**
- Synchronous (fixed per instrumentation)
  - **trace**

Asynchronous Experiments - Profiling

- **profil**, **mprofil**
- OS captures program counter (PC) every 10 milliseconds
- Lowest overhead for an instrumented program
- Creates most compact experiment data file
- Default asynchronous experiment when no explicit choice is made
- HWPC event values and thread detail are not available

Asynchronous Experiments - Sampling

- **samp_*XX*_time**, **samp_*XX*_ovfl**
- Implemented in user-domain
- Timer interrupt occurs every 10 milliseconds
- HWPC event can be set up to overflow, causing interrupt (default is cycles @ 1 ms)
- The PC is captured at each interrupt
- Additional data can also be captured
  - Call stack, HWPC event values, dynamic heap state, resource usage state

Synchronous Experiments – Tracing

- **trace**
- Counts number of times an entry point was called
- Timestamps, function argument values, return values, call stack, HWPC event values
- User-defined entry points (Fortran, C, C++)
- Predefined “trace function groups”
  - MPI, SHMEM, CAF, UPC, Pthreads, OpenMP, Dynamic Heap, System Calls, IO

Instrumenting a Program

- No recompilation is required
- **pat_build** instruments an executable program with a single link
- By default, the program is instrumented for an asynchronous experiment - profiling
- Can change type of asynchronous experiment and data collection at run-time
- Choose entry points for tracing at this time (results in synchronous experiment)

Executing the Instrumented Program

The 45th CUG Conference
CRAY USER GROUP
FLIGHT TO INSIGHT

- Environment variables allow control over various run-time features (i.e., experiment, rate, call stack)
- No need to re-instrument the program to effect different data collection
- Choose the type of asynchronous experiment to change the default of profiling
- Choose the type of data to collect
- Execute instrumented program just as the original
- Creates a binary experiment data file

Evaluating the Results

- **pat_report** analyzes data and displays a formatted textual report
- Control over how and what data is displayed
- Control over appearance of report
- Export data to XML or spreadsheet formats
- Time spent at function, block, source line
- Call trees (caller-callee relationships)
- Per-process, per-thread, per-SSP granularity
- HWPC event values

Application Programming Interface

- Fortran, C, C++
- Allows finer control over data collection
- Collects same information as tracing
- Optionally collects user-specific values
- Allows for turning data recording off and on
- Facilitates tracing of certain programming constructs (e.g., inlined functions, assembly language routines, loops)
- API calls added to original program
- API calls only active in instrumented program

Application Programming Interface

```
use pat_api
. . .
      call PAT_trace_user( "MXM" )
      DO 120 II = 1, IT
        CALL MXM (A, B, C, L, M, N)
120     CONTINUE
      call PAT_trace_user( "" )
. . .
```

API Example

MFLOPS	P:0:0 SSP	Avg VL	Function
7140.87	23496076	64.00	MXM
1792.22	23404284	64.00	ssp. 0
1801.11	23288721	64.00	ssp. 1
1787.55	23465360	64.00	ssp. 2
1785.22	23496076	64.00	ssp. 3

API Example (cont'd)

MFLOPS	P:0:0 SSP	Avg VL	Function
1620.50	78313597	29.04	BTRIX
829.83	77497992	28.55	ssp. 0
266.44	78313597	29.49	ssp. 1
266.81	78203738	29.49	ssp. 2
268.21	77796118	29.49	ssp. 3

API Example (cont'd)

MFLOPS	P:0:0 SSP	Avg VL	Function
761.52	127927386	56.37	GMTRY
580.09	127927386	57.84	ssp.0
96.69	80004853	49.43	ssp.1
94.69	81694289	49.43	ssp.2
99.04	78131379	49.43	ssp.3

Coding Style Example

```
 MFLOPS  |     FLOPs | Avg.VL  |Function
   68.76  |319243515 |   54.09  |Total
|-----------------------------------
|  503.54  |  47070782 |    55.90  |count3_
|  491.14  |  47017686 |    55.90  |count2_
|  480.29  |  46495370 |    39.08  |count5_
|  395.82  |  46495250 |    61.38  |count4_
|  342.03  |  56931216 |    56.55  |count1_
```

Coding Style Example (cont'd)

```
     FLOPS   | MFLOPs | Avg.VL |Function
                             |Arg#2
|| 11900617 | 522.18 |   55.83 |count3_
|||-----------------------------------
|||   2623744 | 563.46 |   63.97 |(128)
|||   2309260 | 575.23 |   60.97 |(122)
|||   1953232 | 563.86 |   57.97 |(116)
|||   1665388 | 510.29 |   54.47 |(109)
|||   1354440 | 498.71 |   50.97 |(102)
|||    975088 | 432.93 |   45.97 |(92)
|||    690505 | 413.19 |   40.47 |(81)
|||    328960 | 437.22 |   63.79 |(64)
```

IO Example

```
 Time% |        Time | Traces |  Input |Output|Function

100.0% | 310.319100 |     555 |1648240 |    3106 |Total
|-------------------------------------------------
| 99.1% | 307.470239 |       1 |      -- |      -- |main
|   0.6% |    2.013595 |     403 |1648240 |      -- |read
|   0.3% |    0.813111 |     150 |      -- |   3106 |write
|   0.0% |    0.019983 |       0 |      -- |      -- |(N/A)
|   0.0% |    0.002172 |       1 |      -- |      -- |_exit
|=================================================
   =====
```

IO Example (cont'd)

```
 Time  |  Traces  |Function
                   |Arg#1:Return

|     2.013595 |      403  |read
||--------------------------------
||      2.009012 |       402  |(0):4096
||      0.004583 |          1  |(0):1648
||================================
|     0.813111 |      150  |write
||----------------------------------
||      0.214141 |        27  |(2):10
||      0.116898 |        28  |(2):4
||      0.089263 |        17  |(2):8
||      0.049679 |        10  |(1):69
```

IO Example (cont'd)

```
|      553  |main
||-----------------------
||      405  |arch_init
||           |  readpackfile
||||---------------------
||||      399  |fscanf
||||           | _doscan
||||||-------------------
||||||      296  |number
||||||           | _filbuf
||||||           |  read
||||||      103  |_filbuf
||||||           | read
```

IO Example (cont'd)

```
|       403  |read
|              | _filbuf
|||------------------------
|||      296  |number
|||              | _doscan
|||             |  fscanf
|||              |    readpackfile
|||             |      arch_init
|||            |        main
|||      107  |_doscan
...
```

DM Example

```
 Samp% | Cum.Samp% | Samp |Calltree#1
                    |Calltree#2
...
|| 17.6% |     17.6% |  104 |Init_Buffers
|| 12.2% |     29.8% |   72 |Exchange
||  8.8% |     38.6% |   52 |Allgatherv
||  8.3% |     46.9% |   49 |PingPong
||  7.1% |     54.0% |   42 |PingPing
||  6.9% |     60.9% |   41 |Sendrecv
||  6.9% |     67.9% |   41 |Allgather
||  6.4% |     74.3% |   38 |Reduce_scatter
||  6.4% |     80.7% |   38 |Allreduce
||  5.6% |     86.3% |   33 |Alltoall
||  3.7% |     90.0% |   22 |Output
||  3.2% |     93.2% |   19 |Bcast
||  2.9% |     96.1% |   17 |Reduce
```

DM Example (cont'd)

```
|| 12.5% |    74 |Exchange
|||-----------------------------------
||| 56.8% |    42 |MPI_Recv
||||----------------------------------
|||| 97.6% |    41 |MPI_CRAY_recv_wait
|||||---------------------------------
||||| 51.2% |    21 |MPI_CRAY_recv_wait(exclusive)
||||| 36.6% |    15 |bcopy
|||||       |       | __bcopy_wordstrm
||||| 12.2% |     5 |__bcopy_prv
|||||=================================
||||  2.4% |     1 |MPI_Recv(exclusive)
```

DM Example (cont'd)

```
|  11.3% |     67 |MPI_CRAY_recv_wait
||-------------------------------
||  61.2% |     41 |pe.0
||  38.8% |     26 |pe.1
||================================
|   7.3% |     43 |MPI_CRAY_reduce
||-------------------------------
||  18.6% |      8 |pe.0
||  81.4% |     35 |pe.1
```

Compare CrayPat to Cray PVP Tools

- ATexpert – tracing (**trace**) OpenMP, Pthreads, system calls
- Flowtrace – **samp_cs_time** and all tracing (**trace**) experiments
- **hpm – pat_hwpc** utility
- Jumptrace – tracing (**trace**) user functions
- Perftrace – **samp_pc_time** and all tracing (**trace**) experiments capturing HWPC event values
- **procstat** – **samp_ru_time**, tracing (**trace**) IO, heap, system calls
- **prof** – **profil**, **mprofil**, **samp_pc_time**

Compare CrayPat to Cray MPP Tools

- MPP apprentice – tracing (**trace**) MPI, SHMEM, UPC, CAF
- MPP pat – **profil**, **mprofil**, **samp_pc_time**, most tracing (**trace**) experiments

Planned Improvements and Features

- Optimize the run-time library
- Optimize HWPC event values collection
- Collect new data (e.g., no forward progress)
- Reduce experiment data file size
- Enhance and optimize report analysis
- Implement GUI and underlying PerfShell (interactive character-based interface)

Conclusion

- CrayPat provides performance analysis from a single point of entry
- Performance evaluated on a per-process, per-thread, and per-SSP level
- No recompilation of program required
- API enables reduction of data collected and focused analysis
- **pat_help** utility offers some on-line help
- Performance API (PAPI) from U of Tennessee available June 2003

Conclusion (cont'd)

- Development continues

Steve Kaufmann

sbk@cray.com

Bill Homer

homer@cray.com

Appendix

The following are slides that supplement this presentation with additional details and are available for viewing provided enough time.

Application Programming Interface

- PAT_profiling_state (int s)
- PAT_sampling_state (int s)
- PAT_tracing_state (int s)
- PAT_record_ssp (int s[ ])
- PAT_trace_user_l (const char *pid, int expr, ...)
- PAT_trace_user_v (const char *pid, int expr, int nargs, long *args)
- PAT_trace_user (const char *pid)
- PAT_trace_function (const void *addr, int cmd)

Run-time Environment Variables

- Commonly use run-time environment variables
  - PAT_RT_EXPERIMENT
  - PAT_RT_RATE
  - PAT_RT_HWPC
  - PAT_RT_HWPC_OVERFLOW
  - PAT_RT_FUNCTION_LIMITS
  - PAT_RT_FUNCTION_MAX
  - PAT_RT_RECORD_SSP

Instrument a Program – pat_build

- Synopsis

pat_build [-d *dirfile*]

[-D *directive*] [-f] [-n]

[-t *tracefile*] [-T *tracefunc*] [-v]

[-w] [-z] *program instr_program*

Evaluating the Results – pat_report

- Synopsis

pat_report [-c stats|records]
[-f txt|xml] [-i *instrprog*]
[-o *output_file*] [-O *options_file*]
[-t *threshold*] [-d *d-opts*]
[-b *b-opts*] [-s *key=value*]
***inst.outPIDem.*xf ...**

API Example

```
=> pat_build -w -T main -f \
   naskern naskern+trace
=> PAT_RT_RECORD_SSP=0-3 \
   PAT_RT_HWPC='P:*:0,P:25:1' \
   aprun naskern+trace
=> pat_report -d mflops,P:0:0,vl \
   -b function,ssp \
   naskern+trace+118777tt.xf
```

Coding Style Example

```
=> pat_build -w \
   -T '/^count/' ising ising+trace
=> PAT_RT_RECORD_SSP=0-3 \
    PAT_RT_HWPC='P:*:0,P:25:1' \
   ising+trace 8
=> pat_report -d mflops,flops,vl \
   -b function \
    ising+trace+5376tt.xf
=> pat_report -d flops,mflops,vl \
   -b ssp,function,argument2 \
    ising+trace+5376tt.xf
```

IO Example

```
=> pat_build \
    -t $PAT_SV2/lib/TraceAIO \
    -t $PAT_SV2/lib/TraceFIO \
    -t $PAT_SV2/lib/TraceIO \
   equake  equake+tio
=> aprun  equake+tio < equake.5
=> pat_report  -d  time%,time,traces,io \
   -b function  equake+tio+58269t.xf \
   > equake.rpt1
```

IO Example (cont'd)

```
=> pat_report -d time,traces \
   -b function,ar1:return ...
=> pat_report -d traces \
   -b calltree ...
=> pat_report -d traces \
   -b function,callers ...
```

DM Example

```
=> pat_build PMB-MPI1 \
   PMB-MPI1+async
=> PAT_RT_EXPERIMENT=samp_cs_time \
   mpirun -np 2 PMB-MPI1+async
=> pat_report -b ct1,ct2 \
   -s percent=relative \
   PMB-MPI1+async+26200sd.xf \
   > bounce.rpt1
```

DM Example (cont'd)

```
=> pat_report -d samp%,samp -b ct \
   -s percent=relative \
   PMB-MPI1+async+26200sd.xf \
   > bounce.rpt2
=> pat_report -d samp%,samp \
   -b fu,pe -s percent=relative \
   PMB-MPI1+async+26200sd.xf \
   > bounce.rpt3
```