BG/Q Performance Tools

Scott Parker

BG/Q Early Science Workshop: March 19-21, 2012

Argonne Leadership Computing Facility

BG/Q Performance Tool Development

- In conjunction with the Early Science program an Early Software efforts was initiated to bring widely used performance tools, debuggers, and libraries to the BG/Q
- Lesson learned and applied to Q:
  - Work with the vendor (IBM) as early as possible in the process:
    - vendor doesn't always have a clear idea of tools/user requirements
    - vendor expertise will dissipate after development phase ends
  - Don't rely solely on vendor tools
  - A number of useful and popular open source HPC tools exist
  - Engage the tools community early for requirements and porting
  - Insure functionality of vendor base software layer supporting tools:
    - hardware performance counter API, stack unwinding, handling interrupts and traps, debugging and symbol table information, debugger process control, compiler instrumentation

BG/Q Tools Development Status

Tool Name	Source	Provides	Q Status
BGPM	IBM	HPC	In testing
gprof	GNU/IBM	Timing (sample)	In development
TAU	Unv. Oregon	Timing (inst), MPI	In testing
Rice HPCToolkit	Rice Unv.	Timing (sample), HPC (sample)	In testing
IBM HPCT	IBM	MPI, HPC	In development
mpiP	LLNL	MPI	In testing
PAPI	UTK	HPC API	In testing
Darshan	ANL	IO	In testing
Open|Speedshop	Krell	Timing (sample), HCP, MPI, IO	In development
Scalasca	Juelich	Timing (inst), MPI	In testing
FPMPI2	UIUC	MPI	Development planned
DynInst	UMD/Wisc/IBM	Binary rewriter	In development
ValGrind	ValGrind/IBM	Memory & Thread Error Check	Development pending
Jumpshot	ANL	MPI	In testing

Aspects of providing tools on BG/Q

- The Good - BG/Q provides a hardware & software environment that supports many standard performance tools:
  - Software:
    - Environment similar to 64 bit PowerPC Linux: Provides standard GNU binutils
    - New performance counter API bgpm
  - Performance Counter Hardware:
    - BG/Q provides 424 64-bit counters in a central node counting unit
    - Counter for all cores, prefetchers, L2 cache, memory, network, message unit, PCIe, DevBus, and CNK events
    - Countable events include: instruction counts, flop counts, cache events, and many more
    - Provides support for hardware threads and counters can be controlled at the core level
- The Bad – some aspects of the hardware/software environment are unique to BG/Q
  - O/S 'Linux like' but not Linux
    - doesn't support all Linux system calls/features: doesn't have fork(), /proc, shell
  - static linking by default
  - non-standard instruction set (addition of quad floating point SIMD inst.)
  - hardware performance counter setup is unique to BG/Q:
    - Some events have node level not core level context
    - Counter setup and operations need to be consistent across cores
    - Some counter operations can have higher latency

Tools and API's Available on Cetus

Tool Name	Source	Provides
BGPM	IBM	HW Counter API
PAPI	UTK	HW Counter API
gprof	GNU/IBM	Timing (sample)
TAU	Unv. Oregon	Timing (inst), MPI, HW Counters (inst)
Rice HPCToolkit	Rice Unv.	Timing (sample), HW Counters (sample)
IBM HPCT	IBM	MPI, HW Counters
mpiP	LLNL	MPI
Darshan	ANL	IO
Jumpshot	ANL	MPI

Performance Tools

- Performance tools collect information during the execution of a program to enable the performance of an application to be understood, documented, and improved
- Different tools collect different information and collect it in different ways
- It is up to the user to determine:
  - what tool to use
  - what information to collect
  - how to interpret the collected information
  - how to change code to improve the performance

Types of Performance Information

- Types of information collected by performance tools:
  - Time in routines and code sections
  - Call counts for routines
  - Call graph information with timing attribution
  - Information about arguments passed to routines:
    - MPI – message size
    - IO – bytes written or read
  - Hardware performance counter data:
    - FLOPS
    - Instruction counts (total, integer, floating point, load/store, branch, …)
    - Cache hits/misses
    - Memory, bytes loaded and stored
    - Pipeline stall cycle
  - Memory usage

Sources of Performance Information

- Instrumentation:
  - Insert calls to performance collection routines into the code to be analyzed
  - Allows a wide variety of data to be collected
  - Source instrumentation can only be used on routines that can be edited and compiled, may not be able to instrument libraries
  - Can be done: manually, by the compiler, or automatically by a parser, or after compilation by a binary instrumenter

Sources of Performance Information

- Sampling:
  - Requires virtually no changes made to program
  - Execution of the program is halted periodically by an interrupt source (usually a timer) and location of the program counter is recorded and optionally call stack is unwound
  - Allows attribution of time (or other interrupt source) to routines and lines of code based on the number of time the program counter at interrupt observed in routine or at line
  - Estimation of time attribution is not exact but with enough samples error is negligible
  - Require debugging information in executable to attribute below routine level
  - Performance data can be collected for entire program including libraries

Sources of Performance Information

- Library interposition:
  - A call made to a function is intercepted by a wrapping performance tool routine
  - Allows information about the intercepted call to captured and recorded, including: timing, call counts, and argument information
  - Can be done for any routine by one of several methods:
    - Some libraries (like MPI) are designed to allow calls to be intercepted. This is done using weak symbols (MPI_Send) and alternate routine names (PMPI_Send)
    - LD_PRELOAD – can force shared libraries to be loaded from alternate locations containing interposing routines, which then call actual routine
    - Linker Wrapping – instruct the linker to resolve all calls to a routine (ex: malloc) to an alternate wrapping routine (ex: malloc_wrap) which can then call the original routine

Sources of Performance Information

- Hardware Performance Counters:
  - Hardware register implemented on the chip that record counts of performance related events. For example:
    - FLOPS - Floating point operations
    - L1 cache miss – number of L1 requests that miss
  - Processors support a limited set of counters and countable events are preset by chip designer
  - Counters capabilities and events differ significantly between processors
  - Need to use an API to configure, start, stop, and read counters
  - Blue Gene/Q:
    - Has 424 performance counter registers for: core, L1, prefetcher, L2 cache, memory, network, message unit, PCIe, DevBus, and CNK events

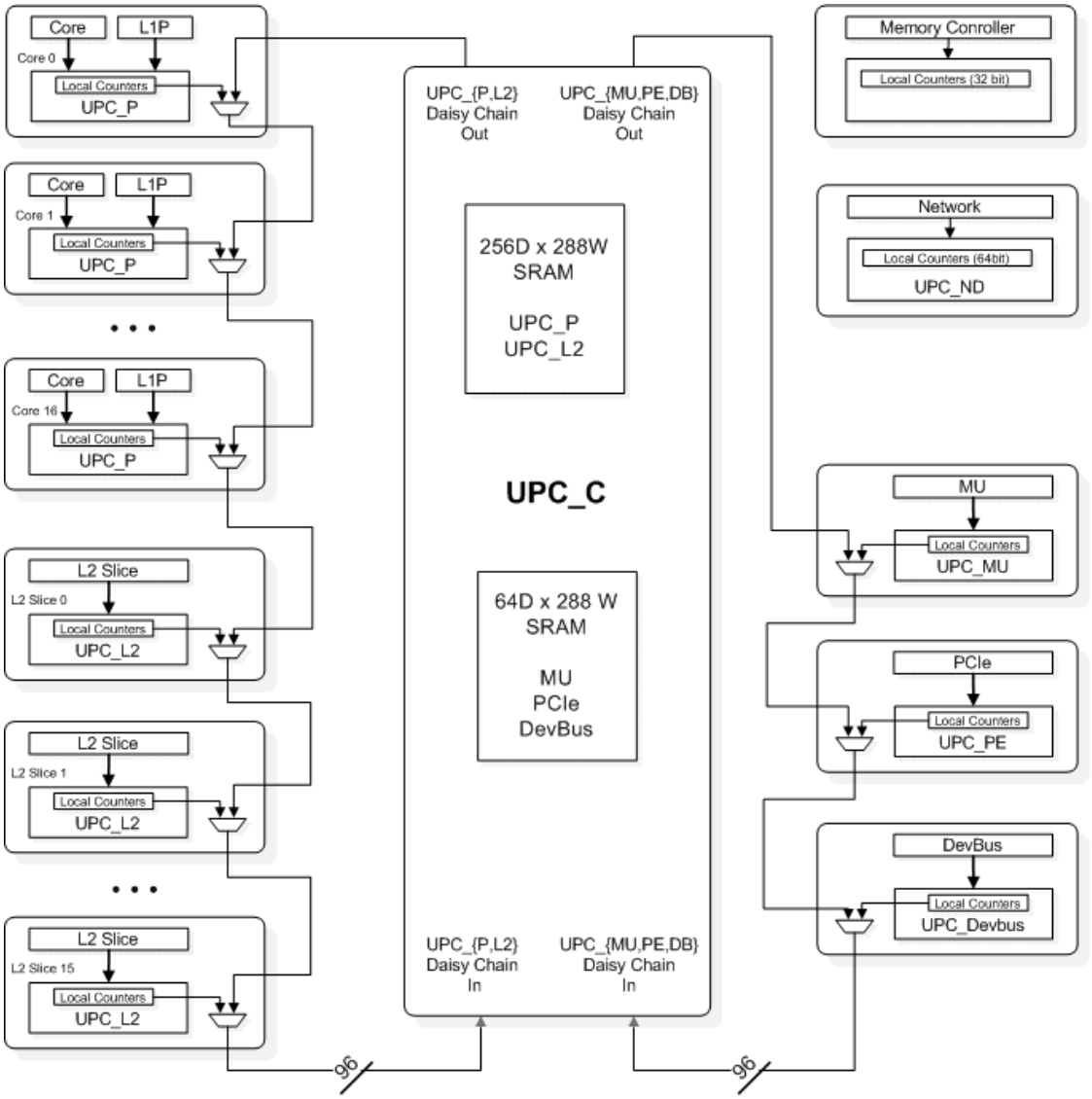
Tools and API's Available on Cetus

Tool Name	Source	Provides
BGPM	IBM	HW Counter API
PAPI	UTK	HW Counter API
gprof	GNU/IBM	Timing (sample)
TAU	Unv. Oregon	Timing (inst), MPI, HW Counters (inst)
Rice HPCToolkit	Rice Unv.	Timing (sample), HW Counters (sample)
IBM HPCT	IBM	MPI, HW Counters
mpiP	LLNL	MPI
Darshan	ANL	IO
Jumpshot	ANL	MPI

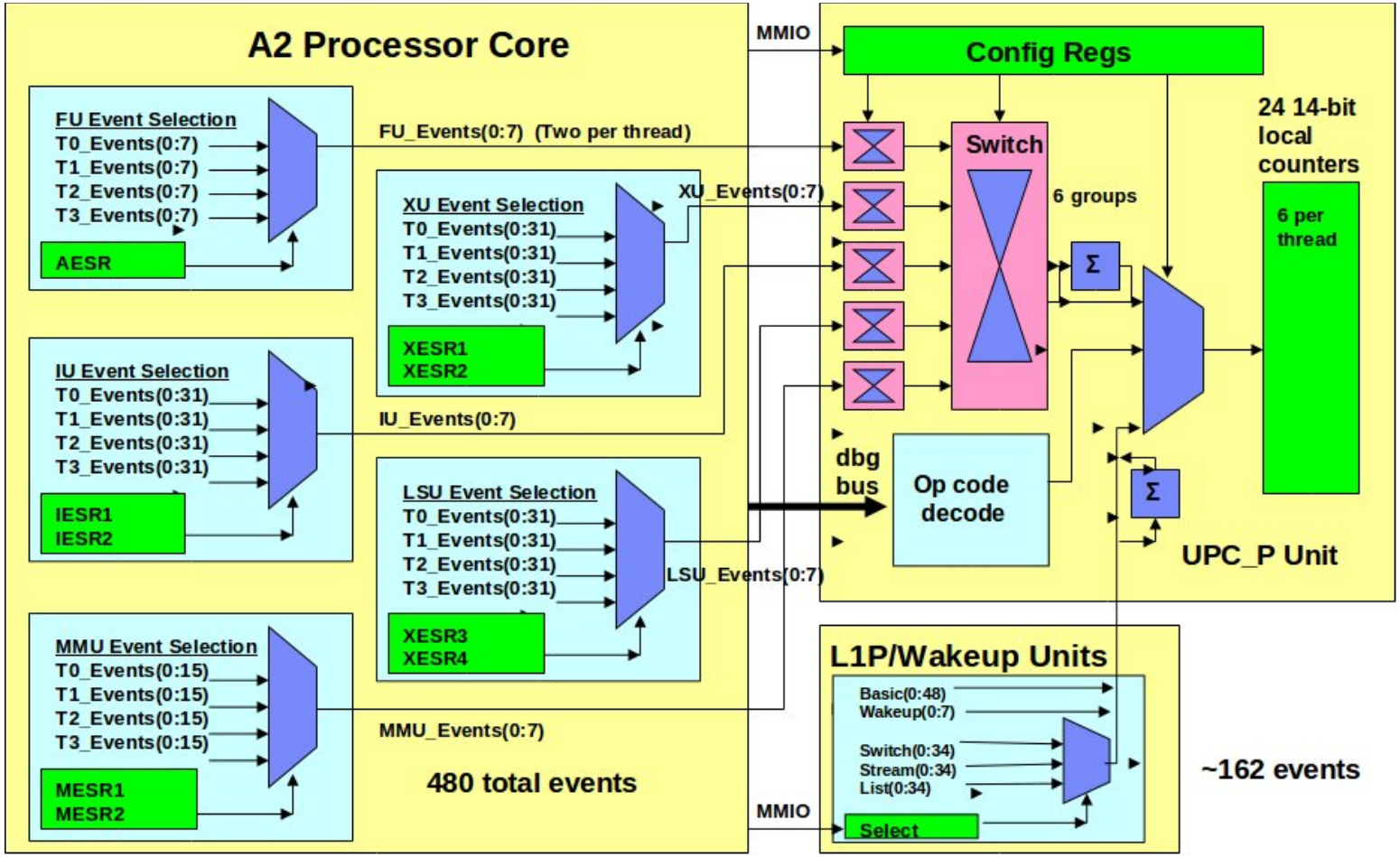
BGPM API

- BGPM is the C programming interface to the hardware performance counters
- Counter Hardware:
  - Centralized set of registers collecting counts from distributed units
    - Sixteen A2 CPU cores, L1P and Wakeup units
    - Sixteen L2 units (memory slices)
    - Message Unit
    - PCIe Unit
    - Devbus
  - Separate counter units for:
    - Memory Controller
    - Network

BGPM Hardware Counters



BGPM Hardware Counters



BGPM API

- Example BGPM calls:

```
Bgpm_Init(BGPM_MODE_SWDISTRIB);
int hEvtSet = Bgpm_CreateEventSet();
unsigned evtList[] = { PEVT_IU_IS1_STALL_CYC, PEVT_IU_IS2_STALL_CYC,
  PEVT_CYCLES, PEVT_INST_ALL, };
Bgpm_AddEventList(hEvtSet, evtList, sizeof(evtList)/sizeof(unsigned) );
Bgpm_Apply(hEvtSet)
Bgpm_Start(hEvtSet);
Workload();
Bgpm_Stop(hEvtSet)
Bgpm_ReadEvent(hEvtSet, i, &cnt);
printf(" 0x%016lx <= %s\n", cnt, Bgpm_GetEventLabel(hEvtSet, i));
```

- Installed in: /bgsys/drivers/ppcfloor/bgpm/
  - Documentation in /bgsys/drivers/ppcfloor/bgpm/docs

BGPM Events

Punit Events

EventId	Label	Description	Scope	Features	Tag	PAPI	Status	Detail
Undefined								
0	PEVT_UNDEF	undefined	thread		s			Undefined event should not occ...
AXU Execution Unit Events (Quad Floating Point Unit)								
1	PEVT_AXU_INSTR_COMMIT	AXU Instruction Committed	thread	olmc	s		v	A valid AXU (non-load/store) i...
2	PEVT_AXU_CR_COMMIT	AXU CR Instruction Committed	thread	olmc	s			A valid AXU CR updater instruc...
3	PEVT_AXU_IDLE	AXU Idle	thread	olmc	s	PAPI_FXU_IDL		No valid AXU instruction is in...
4	PEVT_AXU_FP_DS_ACTIVE	AXU FP Divide or Square root in progress	thread	olmc	be			A Floating-Point Divide or Squ...
5	PEVT_AXU_FP_DS_ACTIVE_CYC	AXU FP Divide or Square root in progress cycles	thread	olmc	bc			Number of Cycles for a Floatin...
6	PEVT_AXU_DENORM_FLUSH	AXU Denormal Operand flush	thread	olmc	s			A B operand of a Floating Poin...
7	PEVT_AXU_UCODE_OPS_COMMIT	AXU uCode Operations Committed	thread	olmc	s		v	A valid AXU ucode operation is...
8	PEVT_AXU_FP_EXCEPT	AXU Floating Point Exception	thread	olmc	e			FP Exception - FX bit of the F...
9	PEVT_AXU_FP_ENAB_EXCEPT	AXU Floating Point Enabled Exception	thread	olmc	e			FP Enabled Exception - FEX bit...

PAPI

- The Performance API (PAPI) specifies a standard API for accessing hardware performance counters available on most modern microprocessors
- PAPI provides two interfaces to counter hardware:
  - simple high level interface
  - more complex low level interface providing more functionality
- Defines two classes of hardware events (run papi_avail):
  - PAPI Preset Events – Standard predefined set of events that are typically found on many CPU's. Derived from one or more native events. (ex: PAP_FP_OPS – count of floating point operations)
  - Native Events – Allows access to all platform hardware counters (ex: PEVT_XU_COMMIT – count of all instructions issued)
- Installed in /soft/libraries/papi-4.1.3.0-bgq-beta-4/

gprof

- Widely available Unix tool for collecting timing and call graph information via sampling
- Collects information on:
  - Approximate time spent in each routine
  - Count of the number times a routine was invoked
  - Call graph information:
    - list of the parent routines that invoke a given routine
    - list of the child routines a given routine invokes
    - estimate of the cumulative time spent in the child routines
- Advantages: widely available, easy to use, robust
- Disadvantages: scalability, opens one file per rank, doesn't work with threads

gprof

- Using gprof:
  - Compile all and link all routines with the '-pg' flag

    ```
    mpixlc -pg -O2 test.c ….
    mpixlf90 -pg -O2 test.f90 …
    ```
  - Run the code as usual: will generate one gmon.out for each rank
  - View data in gmon.out files, by running:

    ```
    gprof <executable-file> gmon.out.<id>
    ```
    - flags:
    - -p : display flat profile
    - -q : display call graph information
    - -l : displays instruction profile at source line level instead of function level
    - -C : display routine names and the execution counts obtained by invocation profiling

TAU

- The TAU (Tuning and Analysis Utilities) Performance System is a portable profiling and tracing toolkit for performance analysis of parallel programs
- TAU gathers performance information while a program executes through instrumentation of functions, methods, basic blocks, and statements via:
  - automatic instrumentation of the code at the source level using the Program Database Toolkit (PDT)
  - automatic instrumentation of the code using the compiler
  - manual instrumentation using the instrumentation API
  - at runtime using library call interception

TAU

- Some of the types of information that TAU can collect:
  - time spent in each routine
  - the number of times a routine was invoked
  - counts from hardware performance counters via PAPI
  - MPI profile information
  - Pthread, OpenMP information
  - memory usage

Rice HPCToolkit

- A performance toolkit that utilizes statistical sampling for measurement and analysis of program performance
- Assembles performance measurements into a call path profile that associates the costs of each function call with its full calling context
- Samples timers and hardware performance counters
  - Time bases profiling is supported on the Blue Gene/Q
- Traces can be generated from a time history of samples
- Viewer provides multiple views of performance data including: top-down, bottom-up, and flat views

IBM HPCT

- A package from IBM that includes several performance tools:
  - MPI Profile and Trace library
  - Hardware Performance Monitor (HPM) API
- Consists of three libraries:
  - libmpitrace.a – provides information on MPI calls and performance
  - libmpihpm.a – provides MPI and hardware counter data
  - libmpihpm_smp.a – provides MPI and hardware counter data for threaded code

- Installed in: /soft/libraries/HPM/
- Documentation in: /soft/libraries/HPM
  - temporarily: /home/sparker/proj/hpm/hpm/doc/MPI_Wrappers_for_BGQ.odt

IBM HPCT – MPI Trace

- MPI Profiling and Tracing library
  - Collects profile and trace information about the use of MPI routines during a programs execution via library interposition
  - Profile information collected:
    - Number of times an MPI routine was called
    - Time spent in each MPI routine
    - Message size histogram
  - Tracing can be enabled and provides a detailed time history of MPI events
  - Point-to-Point communication pattern information can be collected in the form of a communication matrix
  - By default MPI data is collected for the entire run of the application. Can isolate data collection using API calls:
    - summary_start()
    - summary_stop()

IBM HPCT – MPI Trace

- Using the MPI Profiling and Tracing library
  - Recompile the code to include the profiling and tracing library:

```
mpixlc -o test-c test.c -g -L/soft/libraries/HPM -lmpitrace
mpixlf90 -o test-f90 test.f90 -g -L/soft/libraries/HPM -lmpitrace
```

  - Run the code as usual, optionally can control behavior using environment variables.
    - SAVE_ALL_TASKS={yes,no}
    - TRACE_ALL_EVENTS={yes,no}
    - PROFILE_BY_CALL_SITE={yes,no}
    - TRACEBACK_LEVEL={1,2..}
    - TRACE_SEND_PATTERN={yes,no}

IBM HPCT - MPI Trace

- MPI Trace output:
  - Produces files name mpi_profile.<rank>
  - Default output is MPI information for 4 ranks – rank 0, rank with max MPI time, rank with MIN MPI time, rank with MEAN MPI time. Can get output from all ranks by setting environment variables.

```
Data for MPI rank 0 of 1024
Times and statistics from MPI_Init() to MPI_Finalize().
-----------------------------------------------------------------
MPI Routine                        #calls     avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                           3            0.0          0.000
MPI_Comm_rank                           5            0.0          0.000
MPI_Send                             1023          192.0          0.005
MPI_Irecv                            5115          206.4          0.003
MPI_Waitall                             5            0.0          1.668
MPI_Bcast                               3        62125.3          0.001
MPI_Barrier                             3            0.0          0.000
MPI_Reduce                              4         3925.0          0.001
MPI_Allreduce                           5           15.2          0.000
-----------------------------------------------------------------
MPI task 0 of 1024 had the maximum communication time.
total communication time = 1.677 seconds.
total elapsed time       = 38.647 seconds.
heap memory used         = 38.844 Mbytes.
```

IBM HPCT - HPM

- Hardware Performance Monitor (HPM) is an IBM library and API for accessing hardware performance counters
  - Configures, controls, and reads hardware performance counters
- Can select the counter used by setting the environment variable:
  - HPM_GROUP= 0,1,2,3 …
- By default hardware performance data is collected for the entire run of the application
- API provides calls to isolate data collection.

```
void HPM_Start(char *label) - start counting in a block marked by label
void HPM_Stop(char *label) - stop counting in a block marked by label
```

IBM HPCT - HPM

- To Use:
  - Link with HPM library:

```
mpixlc -o test-c test.c -L/soft/libraries/HPM –lmpihpm -L/bgsys/
  drivers/ppcfloor/bgpm/lib –lbgpm –L/bgsys/drivers/ppcfloor/spi/
  lib –lSPI_upci_cnk

mpixlf90 -o test-f test.f90 -L/soft/libraries/HPM –lmpihpm -L/
  bgsys/drivers/ppcfloor/bgpm/lib –lbgpm –L/bgsys/drivers/
  ppcfloor/spi/lib –lSPI_upci_cnk
```

  - Run the code as usual, setting environments variables as needed

IBM HPCT - HPM

- HPM output:
  - files output at program termination are:
    - hpm_summary.<rank> – hpm files containing counter data, summed over node

```
=====================================================================
Hardware counter report for BGQ  - sum for node <0,0,0,0,0>.
cores in use = 16, active threads per core = 4.
=====================================================================
---------------------------------------------------------------------
mpiAll, call count = 1, avg cycles = 61829628876, max cycles = 61835872069 :
  -- Counter values summed over processes on this node ----
0      16279923298   Committed Load Misses
0     146867542314   Committed Cacheable Loads
0      15953740380   L1p miss
0     476269353705   All XU Instruction Completions
0     224066635616   All AXU Instruction Completions
0     444560789965   FP Operations Group 1
  -- L2 counters (shared for the node) -----------------
100    61653146465  L2 Hits
100        383554  L2 Misses
100        190738  L2 lines loaded from main memory
100        368363  L2 lines stored to   main memory


Derived metrics for code block "mpiAll" averaged over process(es) on node <0,0,0,0,0>:
Instruction mix:  FPU = 31.99 %,  FXU = 68.01 %
Instructions per cycle completed per core = 0.7079
Per cent of max issue rate per core = 48.14 %
Total weighted GFlops for this node = 11.503
Loads that hit in L1 d-cache =  88.92 %
                  L1P buffer =   0.22 %
                  L2 cache   =  10.86 %
                  DDR        =   0.00 %
DDR traffic for the node: ld = 0.000, st = 0.001, total = 0.001 (Bytes/cycle)
```

mpiP

- mpiP is a lightweight profiling library for MPI applications
- mpiP provides MPI information broken down by program call site:
  - the time spent in various MPI routines
  - the number of time an MPI routine was called
  - information on message sizes

- Installed in: /soft/perftools/mpiP
- Documentation at: http://mpip.sourceforge.net/

mpiP

- Using mpiP:
  - Recompile the code to include the mpiP library

```
mpixlc -o test-c test.c -L/soft/perftools/mpiP/lib/ -lmpiP -L /bgsys/drivers/
ppcfloor/gnu-linux/powerpc64-unknown-linux-gnu/powerpc64-bgq-linux/ -lbfd -
liberty

mpixlf90 -o test-f90 test.f90 -L/soft/perftools/mpiP/lib/ -lmpiP -L/bgsys/
drivers/ppcfloor/gnu-linux/powerpc64-unknown-linux-gnu/powerpc64-bgq-linux/ -
lbfd –liberty
```

  - Run the code as usual, optionally can control behavior using environment variable MPIP
  - Output is a single .mpiP summary file containing MPI information for all ranks

Darshan

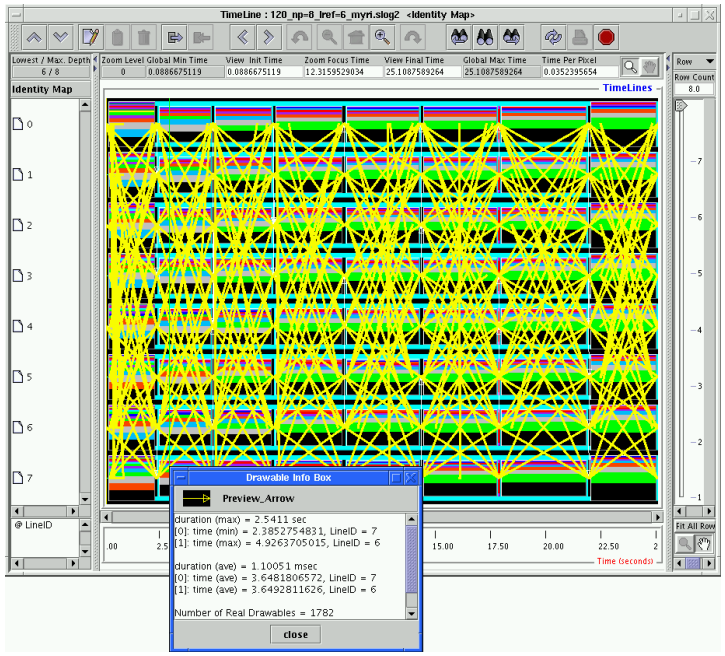
- Darshan is a library that collects information on a programs IO operations and performance
- Unlike BG/P Darshan is not enabled by default yet
- To use Darshan add the appropriate softenv keys:
  - +darshan-xl
  - +darshan-gcc
  - +darshan-xl.ndebug
- Upon successful job completion Darshan data is written to a file named

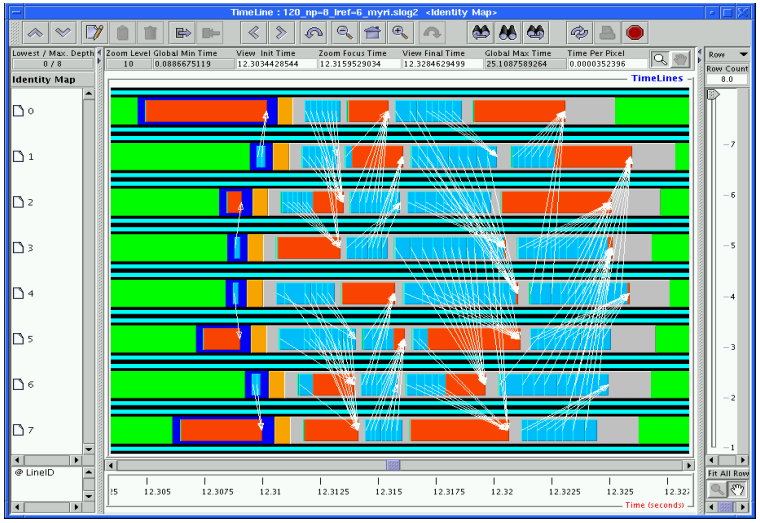
  <USERNAME>_<BINARY_NAME>_<COBALT_JOB_ID>_<DATE>.darshan.gz

  in the directory:
  - /veas-fs0/logs/darshan/<year>/<month>/<day>

Jumpshot

- Jumpshot is an interactive tool for visualizing parallel program behavior
- It can be used to acquire understanding of what a program is doing and why
- It can be used as one of multiple ways to process TAU data
- Events can be inspected at multiple scales





Steps in looking at performance

- Time based profile with at least one of the following:
  - gprof
  - Rice HPCToolkit
  - TAU
- MPI profile with:
  - IBM HPCT MPI Profiling Library
  - mpiP
  - TAU
  - Jumpshot
- Gather performance counter information for critical routines:
  - PAPI
  - HPM