Argonne NATIONAL LABORATORY

# Tools on the Blue Gene P & Q

Scott Parker

U.S. DEPARTMENT OF ENERGY

# Argonne Leadership Computing Facility

- *Intrepid* – IBM Blue Gene/P System:
  - Hardware:
    - PowerPC 450 with 4 cores/node running at 850 MHz
      - Double FPU – 2 wide double precision SIMD
    - 40,960 nodes / 163,840 PPC cores -> 557 Teraflops peak
    - 512 MB per core
  - Software:
    - Compute Node Kernel (CNK) O/S (Linux like)
    - Static linking by default
    - Cross compiling required due to different login/compute environments

- *Mira* – IBM Blue Gene/Q System
  - Next evolution of the Blue Gene architecture
  - Hardware:
    - PowerPC A2 with 16 cores/node running at 1.6 GHz
      - 4 way multi-threaded
      - Quad FPU – 4 wide double precision SIMD
      - Transactional memory, speculative execution, list prefetch
    - Over 780k cores -> over 10 Petaflops peak
      - Possible to run with over 3 million processes/threads
    - 1 GB of memory per core
  - Software:
    - Generally similar to BG/P

# ALCF Tools Status

| Tool Name | Source | Provides | P Status | Q Status |
|---|---|---|---|---|
| gprof | GNU/IBM | Timing (sample) | Supported | In development |
| TAU | Unv. Oregon | Timing (inst), MPI | Supported | Development pending |
| Rice HPCToolkit | Rice Unv. | Timing (sample), HPC (sample) | Supported | In development |
| IBM HPCT | IBM | MPI, HPC | Supported | In development |
| mpiP | LLNL | MPI | Supported | Development uncertain |
| PAPI | UTK | HPC API | Supported | In development |
| HPM | IBM | HPC API | Supported | Development uncertain |
| Darshan | ANL | IO | Supported | Development pending |
| Open\|Speedshop | Krell | Timing (sample), HCP, MPI, IO | Beta | Development pending |
| Scalasca | Juelich | Timing (inst), MPI | Not Installed | Development planned |
| FPMPI2 | UIUC | MPI | Not Installed | Development planned |
| TotalView | Rouge Wave | Debugger | Supported | Development planned |
| DDT | Allinea | Debugger | Supported | Development planned |
| Coreproccessor | IBM | Debugger/Stack-trace | Supported | In development |
| DynInst | UMD/Wisc/IBM | Binary rewriter | Not Installed | Development planned |
| ValGrind | Valgrind/IBM | Memory & thread error check | Not Available | Development planned |

# Issues in providing tools on BG/P

- Vendor tools are extremely limited
- Heavily dependent on open source tools
- Numerous porting and functionality issues:
  - O/S 'Linux like' but not Linux
    - doesn't support all Linux system calls/features:
      - fork(), /proc, no shell
  - static linking by default
  - non-standard instruction set (addition of dual floating point SIMD inst.)
  - hardware performance counter limitations:
    - Configuration and control is node centric, not core centric
    - can only count on half the cores at a time
    - not all event have core context
    - not documented

# What we learned for BG/Q

- Work with the vendor 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 on vendor tools
- Engage the tools community early for requirements and porting
- Identify and focus on base software layer supporting tools:
  - hardware performance counter API, stack unwinding, interrupts and traps, debugging and symbol table information, debugger process control, compiler instrumentation
- Expect to have to have to assist with architecture expertise and porting issues

# Information Available on BG/P

- Time in routines and code sections (via instrumentation or sampling)
- Call graph information
  - with timing attribution
  - routine call counts
- Information about arguments passed to routines:
  - MPI – message size
  - IO – bytes written or read
- Limited hardware performance counter data:
  - FLOPS
  - ~ Instruction counts (total, integer, floating point, load/store, branch)
  - ~ Cache hits/misses, bytes loaded and stored from memory
- Limited memory usage data
- Debug at moderate scale

# Desired features

- Better hardware performance counters:
  - core/thread context wherever appropriate
  - more counters
  - Better set of events: flops, instruction counts, stall cycles, cache & memory hierarchy
  - more configuration flexibility
- Support for unique architectural features (SMT, transactional memory, prefetcher)
- Thread profiling (OpenMP, pthreads):
  - parallel startup overhead, coverage, synchronization overhead
- Correctness checks:
  - memory corruption
  - threads and race conditions
- Memory usage and layout
- Network Information (congestion)
- OS information (thread migration/scheduling, memory)
- Power usage

# What a ‘typical’ user wants in a tool

- Familiar:
  - the first tool a user will use is the one they know
  - consistent across platforms and generations of platforms
- Well documented
- Robust:
  - first time a tool sends an error message is often the last time they use it
- Easy-to-use:
  - no code changes
  - easy to enable
  - minimal number of runs to collect data
- Produce concise, easily digested information:
  - measured performance
  - bottleneck locations and causes
  - corrective actions

*With every step removed from ideal usage decreases exponentially*
*User don’t necessarily know the specifics of what they want*

# Exascale Tools

- More robust
- Much more scalable
- Better documented
- Simpler
- Smarter (tools or users):
  - Tools: better able to translate collected performance data to performance improvements
    - identify bottlenecks and cause with clear attribution to code sections
    - suggest remedies
  - Users: better educated, need to know what to do with the information the tools provide
- Provide the features listed previously
- *Tool developers should be involved from platform inception, through development, deployment, and into operation*
- *Commonality*