Performance Tools on BlueGene/L

or

“So, how well am I using those 64K nodes, anyway?”

Philip C. Roth    Barton P. Miller

{pcroth,bart}@cs.wisc.edu
Paradyn Project
Computer Sciences Department
University of Wisconsin-Madison

Madison, WI 53706 USA
Challenges for Performance Tools on BlueGene/L

- Flexible, non-intrusive generation of performance data on compute nodes
- Non-intrusive transfer of performance data off compute nodes
- Scalable analysis of performance data
- Scalable visualization of performance data and analysis results
- Mechanism for launching application under tool control, attaching tool to running application
Generating Performance Data

• Static instrumentation (i.e., source-based or link-time instrumentation, binary rewriting) is restrictive on BG/L
  - Limited hardware availability—system time is precious, static instrumentation is not adaptive
  - Limited address space per node, wasted by unused instrumentation code

• Better: use Dynamic Instrumentation
  - Add/remove instrumentation code in running processes
  - Adaptive—change what’s being collected dynamically
  - Remove unused instrumentation code
Implementing Dynamic Instrumentation on BlueGene/L

- **On traditional platforms**
  - ptrace/procfs
  - no pre-execution preparation needed

- **Possible approaches for BG/L:**
  - Debugger interface via CIO daemons
    - Latency for making process changes?
    - Ability to pause processes asynchronously?
  - In-process run-time support code
    - Injected into process via debugger interface, or linked into executable
    - Control transfer to run-time code triggered by asynchronous receipt of instrumentation request?
Transferring Performance Data from Compute Nodes

- **Goal:** minimize application contention for CPU and for network(s)

- **Some approaches on BG/L:**
  - Push data when generated using tree or control network (low-volume traces, low-rate sampling)
  - Buffer data (memory or SRAM scratchpad)
    - Push data when buffer fills
    - Push data on performance counter “interrupt”
    - Pull data using JTAG network (SRAM, HW counters)

- Best approach depends on rate of data generation and intra-node storage requirements
Scalable Performance Data Analysis

• On-line analysis is often desirable
  - Avoids cost of storing large performance data sets for post-mortem analysis
  - Required to support dynamic adaptation, e.g. automated tools, adaptive communication layers

• Possible approaches on BG/L:
  - Analyze all data on host system(s)
  - Use distributed analysis in tool daemons on I/O nodes
  - Leverage second core?
Example: Paradyn

• Features
  - Performance Consultant for automatically finding application performance problems
  - Dynamic instrumentation for on-line, flexible performance data collection
  - MRNet infrastructure for scalability

• Data rate is variable; tens of KB/s per application process is typical
Paradyn: Logical Tool Organization

Multicast/Reduction Network

Front End

Internal Process

Filter

Tool Daemons

d_0  d_1  d_2  d_3  ...  d_{P-4}  d_{P-3}  d_{P-2}  d_{P-1}

App Processes

a_0  a_1  a_2  a_3  ...  a_{P-4}  a_{P-3}  a_{P-2}  a_{P-1}
Paradyn on BlueGene/L

- Global tool control in front-end on host system
- Increasingly aggressive approaches for dynamic instrumentation, data collection and analysis:
  1. Tool daemons on host system; no MRNet; use CIO daemon debugger interface
     - Gain experience with dynamic instrumentation on compute nodes
     - Suitable for small BG/L configurations only
  2. Tool daemons on I/O nodes; no or small MRNet on host system; control dynamic instrumentation run-time support code over tree network
     - Scalable search control, non-scalable global data aggregation
     - Custom interface required over tree network
  3. MRNet processes on I/O nodes; use tree network reductions when possible
  4. Leverage compute nodes for analysis
http://www.paradyn.org
http://www.dyninst.org
paradyn@cs.wisc.edu