Tracing for Performance Monitoring on Parallel and Distributed Systems

Dr. Robert W. Wisniewski
Manager Blue Gene Software
IBM T. J. Watson Research
http://www.research.ibm.com/people/b/bob/
Outline

- SGI (RTAS – Real-Time Technology and Applications Symposium 95)
  - rtmon
  - Kernel and Cray Unification
  - Lessons
- K42 (Supercomputing 03)
  - Approach, scalability, and use
  - Lessons
- CPO (Continuous Program Optimization) (PAC2 2004)
  - PEM (Performance Environment Monitoring)
  - Lessons
- CSO (Commercial Scale-Out) (europar07 – slides thanks Jose Moreira)
  - Goals
  - Lessons
- Blue Gene / P (internal – slides thanks Valentina Salapura)
- Observations on Linux and LTT
- The “next system” - concluding remarks
Frame Scheduler

Major Frames: Determines period - a complete cycle of processes

Minor Frames: Independent units within major frame - used for setting up specific application behavior

Processes can be enqueued in more than one process queue
SGI rtmon

- rtmon bottom layer
  - Per-processor
  - Multiple writes user q
    - atomicIncWrap gets index
    - Set valid bit when done
  - Reader clears valid bit

FrameView - Bottom Layer

Library Calls for accessing merged stream of data:

- rtmon_get_next_event
- rtmon_get_time
- rtmon_get_pid
- rtmon_get_type
- rtmon_pause_kern
- rtmon_resume_both

typedef struct {
  long long time;
  int event;
  int pid;
  int type;
} merge_event_t;
SGI rtmon

- rtmon middle layer
  - Assign meaning to events, recreate frames
  - Report discrepancies
  - Calculate extreme value

FrameView - Middle Layer

Library Calls for accessing major frame statistics
- rtmon_get_next_ave
- rtmon_get_startup_times
- rtmon_get_frame_ave
- rtmon_get_maj_proc_ave
- rtmon_set_frame_ave_factor
- rtmon_set_proc_ave_factor
- rtmon_get_max_time
- rtmon_get_max_kern_time
- rtmon_get_max_proc_time

Monitoring Distributed Systems for Diagnostic Purposes © 2008 IBM Corporation
SGI rtmon

- rtmon top layer
  - Multiple view

Kernel Startup Graph
Real-Time Monitoring Tools

Above represents full view matching the view as seen in the main graph
Below represents a blown up image of the kernel startup time for minor frame 0

Colors of event labels match color bars
SGI Kernel and Cray Unification

- rtmon extended to kernel
  - 3 separate tracing schemes depending on what you were doing
    - Confusing
    - Error prone
    - Hurts performance

- SGI purchases Cray
  - 5 separate tracing schemes...
  - Cray introduces another aspect
    - Need data from machines in field that are not possible to build in house – requires extensive events and black-box capability
SGI Lessons

- **rtmon**
  - Collect cheaply on line more expensive off line processing
  - Roughly ¼ of machine needed to get events off
  - Tradeoff between application-specific design and generality
  - Single system of trace events useful
  - Possible to do non-locking tracing
  - Fixed events are cumbersome

- Visualization is key
  - It’s the killer app for tracing
Outline

- SGI
  - rtmon
  - Kernel and Cray Unification
  - Lessons

- K42
  - Approach, scalability, and use
  - Lessons

- CPO (Continuous Program Optimization)
  - PEM (Performance Environment Monitoring)
  - Lessons

- CSO (Commercial Scale-Out)
  - Goals
  - Lessons

- Blue Gene / P

- Observations on Linux and LTT

- The “next system” - concluding remarks
K42’s Goals (started 1997)

- **Scalability**
  - Up to large MP and large applications
  - Down for small-scale MP and small apps on large-scale MP

- **Flexibility/Customizability:**
  - Policies/implementations of every physical/virtual resource instance can be customized to application needs
  - System can adapt to security and performance faults without penalizing common case performance

- **Portability:**
  - Can be easily ported to new 64-bit platforms
  - Can exploit features of HW

- **Availability:**
  - Fault containment: should be able to survive HW failures on large MP
  - Can be dynamically upgraded without bringing system or apps down

- **Maintainability/Extensibility:**
  - Highly module structure
  - Re-enable the OS research community

- **Full Functionality and Linux compatibility:**
  - Support huge numbers of Linux apps and drivers without modification
  - Transfer technology back and forth to vanilla Linux
Goals: Performance Monitoring

- Provide unified events for correctness and performance
- Allow events to be gathered efficiently on a multiprocessor
- Allow efficient logging of events from applications, servers, and the kernel into a unified buffer with monotonically increasing timestamps
- Have the infrastructure always compiled into the system allowing data gathering to be dynamically enabled
- Separate the collection of events from their analysis
- Have minimal impact on the system when tracing is not enabled; allow for zero impact by providing the ability to "compile out" events
- Provide cheap and flexible collection of data for either small or large amounts of data per event
Key Ideas

* lockless logging
* random access variable length events
  → unified events
  ^ user-mapped per-processor buffers
  ^ major and minor ids
Lockless Logging

0

current index

Process A – to log 2 words
Process B – to log 3 words
Lockless Logging

0

current index

1

proc A

current index

proc B
Lockless Logging

0

current index

1

proc A

proc B

2

current index

current index

B
Lockless Logging

works between user, servers, kernel
potential problems – event loss etc.
Random Access
Variable Length Events

- Variable length events (vs fixed length)
  - more flexible
  - cheaper
    - space
    - time
  - easier for longer events
Lockless Logging

works for RAM and disk
Use Event Listing

21.4747350 TRC_USER_RUN_UL_LOADER
21.4747422 TRC_EXCEPTION_PGFLT
21.4747882 TRC_EXCEPTION_PGFLT_DONE
21.4748091 TRC_EXCEPTION_PPC_CALL
21.4748530 TRC_MEM_FCMCOM_ATCH_REG
21.4748709 TRC_MEM_FCMCRW_CREATE
21.4749142 TRC_EXCEPTION_PPC_RETURN
21.4749247 TRC_EXCEPTION_PPC_CALL
21.4749573 TRC_MEM_REG_CREATE_FIX
21.4749773 TRC_MEM_REG_DEF_INITFIXED
21.4749873 TRC_MEM_ALLOC_REG_HOLD
21.4749962 TRC_MEM_ALLOC_REG_HOLD
21.4750293 TRC_MEM_FCMCOM_ATCH_REG

process 6 created new process with id 7 name /shellServe
PGFLT, kernel thread 80000000c12b0f90, faultAddr 405e628,
PGFLT DONE, kernel thread 80000000c12b0f90, faultAddr 405
PPC CALL, commID 0
Region 800000001022cc98 attached to FCM e100000000003f90
TRC_MEM_FCMCRW_CREATE ref e100000000003f90
PPC RETURN, commID 60000000
PPC CALL, commID 0
Region default 10000000 created fixlen addr 113000
region default init fixed 80000000102b7c00 addr 10000000
alloc region holder addr 10000000 size 113000
alloc region holder addr 10000000 size 113000
Region e100000000003fa0 attached to FCM e100000000003f90
Use Fine-Grained Behavior

pid: 3d parent: 30 lpid: 163 lparent: 157
Exec:./runtest.sh /bin/rmdir
SCexecve : 209.59/1/86 f: 273.20/15 p: 691.53/34
SCexit : 13.43/1/9 f: p: 24.19/5
SCmmap : 53.39/4/42 f: p: 199.94/19
SCrmdir : 13.61/1/3 f: p: 53.92/1
dispatcher : 32.71/1/13 f: 87.53/7 p: 9.77/3
user : 1718.56/27/104 f: 1304.87/76 p:
In-process total: 2500.18/434

-----------------------------------------------
cleanup : 929.41/1/5 f: p:
fault : 2804.31/184/186 f: p:
ppc : 1274.52/93/210 f: p:
Ex-process total: 5008.23/401
wall 10800.11/0
-----------------------------------------------
CRT::ForkChildPhase2 255.32/2
DispatcherDefault::AsyncMsgHandler 4.05/3
CRT::ForkWorker 246.10/4
COSMgrObject::CleanupDaemon 185.61/2
MPMsgMgrEnabled::ProcessMsgList 3.56/1
### Use

**Lock Contention Analysis**

top 10 contended locks by time - for full list see traceLockStatsTime

<table>
<thead>
<tr>
<th>time (secs)</th>
<th>count</th>
<th>spin</th>
<th>max time</th>
<th>pid</th>
<th>call chain</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.466320753</td>
<td>1209</td>
<td>188795433</td>
<td>0.012220087</td>
<td>0x1</td>
<td>AllocRegionManager::alloc(unsigned PMallocDefault::pMalloc(unsigned GMalloc::gMalloc()))</td>
</tr>
<tr>
<td>0.684612632</td>
<td>573</td>
<td>37233770</td>
<td>0.007647854</td>
<td>0x0</td>
<td>AllocRegionManager::alloc(unsigned PMallocDefault::pMalloc(unsigned GMalloc::gMalloc()))</td>
</tr>
<tr>
<td>0.104643241</td>
<td>11885</td>
<td>4910595</td>
<td>0.000322320</td>
<td>0x1</td>
<td>PageAllocatorDefault::deallocPages(unsigned PageAllocatorUser::deallocPages(unsigned AllocPool::largeFree(void*,</td>
</tr>
</tbody>
</table>
K42 Lessons

- K42
  + Static trace points valuable
    - More efficient (94 cycles on K42)
    - Modified when code is modified
  + Separate definition files useful
  + Breakdown into major and minor classes useful
  + Variable length events
  + Single unified system for events
  + Dynamic enabling and disabling useful
    - No dynamic events
    - No flexibility at event time
  🌋 Visualization is key
Outline

- SGI
  - rtmon
  - Kernel and Cray Unification
  - Lessons
- K42
  - Approach, scalability, and use
  - Lessons
- CPO (Continuous Program Optimization)
  - PEM (Performance Environment Monitoring)
  - Lessons
- CSO (Commercial Scale-Out)
  - Goals
  - Lessons
- Blue Gene / P
- Observations on Linux and LTT
- The “next system” - concluding remarks
CPO Vision and Potential

load monitoring, loop instrumentation, PMU analysis, feedback directed optimization (FDO) etc.

Application
- multiple threads
- parallel loops
- data structures
- communication

online info
- monitoring, poor performance detection, evaluation of directives
- migrate thread, redistribute loop, page size request, FDO, etc.

CPO online agent

CPO offline agent
- page benefit analysis, aggregation formation, program analysis, trace conversion

CPO data base

CPE Trace
- Visualizer

CPO online agent

unified framework for including work from many groups
CPO Architecture

CPO online agent

- Application
- App Server
- Java VM
- Libraries
- Operating System
- Hypervisor
- Hardware/Simulator

CPO offline agent

- Event Trace
- PE Trace Visualizer

Static compiler

Feedback-directed optimization

PEM

Continuous monitoring

Online loop

Offline loop

CPO database

Persistent storage of analysis/optimization directives

Control

Data/code

History, optimization directives, trace analysis, modeling
Overview of cth performance using PE
Comparison of large page mapping categories shown in PE
CPO Implementation

- Extended K42’s infrastructure
  - Events from a wider range of layers
    - Extended notion of majors and minors to layers
  - Integrated HW performance counters
  - Self describing event definitions in XML
  - Extended to more than tracing, at each “event”:
    - Trace event
    - Gather statistics on event, with tracing at threshold
    - Call a handler for event
    - All of the above
CPO Lessons

- CPO
  + Vertical integration with HPCs powerful
  + Addition of statistics option good for online monitoring
  + Multiplexing hardware counters (ICS 05)
    - No dynamic events
    - No automatic packaging of trace and description files
  ☀ Visualization was valuable
Outline

- SGI
  - rtmon
  - Kernel and Cray Unification
  - Lessons
- K42
  - Approach, scalability, and use
  - Lessons
- CPO (Continuous Program Optimization)
  - PEM (Performance Environment Monitoring)
  - Lessons
- CSO (Commercial Scale-Out)
  - Goals
  - Lessons
- Blue Gene / P
- Observations on Linux and LTT
- The “next system” - concluding remarks
Introduction

- In scientific/technical computing, parallel processing became mainstream in the 80’s
- Since the early 90’s there has been a strong move of commercial computing away from single-processor machines to multi-processor systems, as the latter became more cost efficient
- Two different approaches to multiprocessors:
  - Scale-up: large shared-memory machines
  - Scale-out: clusters of interconnected smaller machines
Introduction

- In scientific/technical computing, parallel processing became mainstream in the 80’s
- Since the early 90’s there has been a strong move of commercial computing away from single-processor machines to multi-processor systems, as the latter became more *cost efficient*
- Two different approaches to multiprocessors:
  - Scale-up: large shared-memory machines
  - Scale-out: clusters of interconnected smaller machines
Scale-up

Scale-out
Commercial Scale Out experimental system
Understanding performance in commercial scale-out

- **Two challenges similar to scientific computing:**
  - Lots of processing elements → lots of trace data: need techniques to limit data and identify important parts
  - Correlate events from different machines → need synchronized time

- **Two challenges unique to commercial:**
  - Complexity of the software stack → hypervisor, operating system, Java, middleware, application
  - Many threads of execution per processing element → multiple threads per process and multiple processes per processor – it is not unusual to see hundreds to thousands of threads per machine!
Starting point

- **Linux Trace Toolkit Next Generation (LTTng):**
  - Extracts information from hypervisor to application
  - Requires instrumentation but it is uniform across layers
  - Low overhead

- **Linux Trace Toolkit Viewer (LTTV):**
  - Merges data collected by each software layer
  - Identifies the producer of each event (node, process, thread)
  - Classifies the execution context (process, trap, interrupt, system call)

- **Enhancements to LTTng:**
  - PowerPC-specific instrumentation
  - Tracing support for Java – addition of *thread branding* (also LTTV)

- **Performance monitoring facility**
  - Uses hardware performance counters in PowerPC
  - Identified bottlenecks through statistical sampling
Stall breakdown

- ~2 billion completing cycles/sec (20% of total 10 billion)
- 6 billion instructions/sec
- Non-stall CPI ($\text{CPI}_C$): 0.34  
  Average for SPECcpu 2000: 0.35
- Average bundle size: 3
CSO Lessons

- CSO
  + Tracing useful
  + HPCs useful
  - Performance monitoring for distributed commercial workloads needs more work
    • Handling many small, in terms of CPU usage, tasks
    • Automatic process branding
    • Inter-machine timer synchronization
    • Automatic idle determination
    • Cross machine logical causality
    • Tree-based causality
    • Selective aggregation of performance data
    • Virtualization
Outline

- SGI
  - rtmon
  - Kernel and Cray Unification
  - Lessons
- K42
  - Approach, scalability, and use
  - Lessons
- CPO (Continuous Program Optimization)
  - PEM (Performance Environment Monitoring)
  - Lessons
- CSO (Commercial Scale-Out)
  - Goals
  - Lessons

- Blue Gene / P
- Observations on Linux and LTT
- The “next system” - concluding remarks
BlueGene/P

**System**
72 Racks, 72x32x32

1 PF/s
144 TB

**Node Card**
(32 chips 4x4x2)
32 compute, 0-1 IO cards

**Rack**
Cabled 8x8x16

13.9 TF/s
2 TB

**Compute Card**
1 chip, 20 DRAMs

435 GF/s
64 GB

**Chip**
4 processors

13.6 GF/s
8 MB EDRAM

13.6 GF/s
2.0 GB DDR2
(4.0GB is an option)
Performance monitoring unit in the Blue Gene/P system

- Implements 256 counters, 64 bits wide
  - 1024 possible counter events
  - Monitors 4 processor cores and FPU, L3, L2, snoop filters, torus and collective network

- Novel architecture
  - Hybrid implementation using SRAM arrays
  - High density, high capacity on-chip performance monitor unit

- Hybrid architecture
  - 12 low order bits of a counter implemented using discrete logic
  - 52 high order bits stored in an SRAM array
  - SRAM state updated at a regular basis under state machine control
  - Configurable input selection and interrupt
  - Interrupt indication when the threshold value is reached
Hybrid PMU architecture

Counter events

Increment

Carry

Counter Address

52 bit increment

Interrupt threshold reg.

Interrupt

FSM

Interrupt Arm

FSM

FSM

FSM

FSM

FSM

FSM

FSM

FSM

FSM
Usage of PMU in BGP

- Opens countless possibilities – some usage examples
  - Analyze the execution profile for different compiler optimizations and infer their effectiveness
  - Conclude on the effectiveness of the various hardware & software settings to determine the optimal configuration
  - Profile and characterize workloads for various modes of operation to achieve maximum performance on multiple cores
Coreprocessor showing program counter on 4 racks
Data and Control Flow of HPCS Toolkit

HPCS Toolkit provides Autonomic Application Performance Capability.

- Intelligent automation of performance evaluation and decision system
- Interactive capability with graphical/visual interface always available, but always optional
Outline

- SGI
  - rtmon
  - Kernel and Cray Unification
  - Lessons
- K42
  - Approach, scalability, and use
  - Lessons
- CPO (Continuous Program Optimization)
  - PEM (Performance Environment Monitoring)
  - Lessons
- CSO (Commercial Scale-Out)
  - Goals
  - Lessons
- Blue Gene / P
  - Observations on Linux and LTT
- The “next system” – concluding remarks
What is right for Linux and How

- Patches versus dynamic points versus markers versus static
- One infrastructure versus many
- Get performance monitoring community active on lkml
- Get nose in tent
Outline

- SGI
  - rtmon
  - Kernel and Cray Unification
  - Lessons
- K42
  - Approach, scalability, and use
  - Lessons
- CPO (Continuous Program Optimization)
  - PEM (Performance Environment Monitoring)
  - Lessons
- CSO (Commercial Scale-Out)
  - Goals
  - Lessons
- Blue Gene / P
- Observations on Linux and LTT
  - The “next system” concluding remarks
The Final Next System

- Efficient, Flexible tracing
  - Single unified space over all layers including HW counters
  - Use static events or event markers
  - Enable system to trace, gather stats, or callback at event
  - Allow additional dynamic events
  - Break into categories and allow dynamic enabling
  - Provide automatic tool for packaging up data and description
  - Timer synchronization built into infrastructure
  - Variable sized events
  - Non-locking and scalable gathering
  - Efficient online gathering for more extensive offline analysis
  - Negligible impact when disabled
The Final Next System

- Configurable visualization
  - Ability to add new graphs and have system save view
  - Pluggable modules to interpret application-specific events
  - Ability to handle massive (100G +) trace data
    - Quick start up
    - Summary and stats information on selectable portion
  - Handle multicore, multiprocessor, and distributed data
  - Handle real-time, scientific, and commercial data
  - Lots of interesting work left to understand commercial systems
  - Nice default views
    - Time-centric time by process, thread-centric view, statistics, histogram, event list