An Introduction to HPC Tools Research

Karen L. Karavanic
Research Scientist
New Mexico Consortium
and
Associate Professor of Computer Science
Portland State University
Subliminal Slide: You want to go to graduate school here...
In the Beginning...

- 1991 I started graduate school at U Wisconsin
- 1993 I started working with the Paradyne Parallel Performance Tools Group

Hmmm, what’s a parallel performance tool??
Parallel Performance Tools

1. Goal: Locate the Bottleneck
   Applications large/long running
   – Profiling

2. Problem: Synchronization
   – MPI applications:
     • one MPI rank waiting to receive a message
     • Barrier: EVERYONE waiting for slow rank to reach barrier
   – Tracing

3. Scaling
   – Intel Paragon: 1-4000 Processors/Nodes !!
   – How to measure all of the nodes??
   – Perturbation: we can no longer measure everything
Paradyn Parallel Performance Tools

• Key insight (Hollingsworth/Miller 1994): *what if we could insert and remove instrumentation on the fly, as the application runs??*

• Dynamic instrumentation

• Automated Performance Diagnosis
  – Define common problems and “hypotheses”
  – Search through the space of all possible problems (“why”) places in the code (“where”) and phases throughout the long run (“when”)

• First Demo: SC Exhibit Hall

• My Dissertation: Using historical data to make this more efficient
1993 to present: Never Boring...

• Threading
  – Scalability Challenges: 1,000s-10,000 of threads
    • How to measure all of the threads for each node??
    • New bottlenecks: locking
    • How to visualize??

• Multicore
  – Scalability Challenges: 2-8x cores/processor
    • How to measure all of the cores for each processor??
    • Perturbation
    • New Bottlenecks: shift to memory
1993 to present: Never Boring...

• Manycore
  – Example: General Purpose GPU Computing
    • 1000s of low power compute cores
    • Used as “accelerators” to CPUs
    • High level directives: how to explain problem to programmer?

• Power/Cooling
  – Great Berkeley Quote: You can’t cause more climate problems powering your data center than you solve with your science
  – Exascale Reality: We cannot generate enough power to simply expand our systems
  – New Challenge for tools: measure/report power/heat
Trinity @LANL

- Architecture: Cray XC30
- Memory capacity: >2 PB of DDR4 DRAM
- Peak performance: >40 PF
- Number of compute nodes: >19,000
- Processor architecture: Intel Haswell & Knights Landing
  - 60 cores/processor
- Parallel file system capacity (usable): >80 PB
- Parallel file system bandwidth (sustained): 1.45 TB/s
- Burst buffer storage capacity (usable): 3.7 PB
- Burst buffer bandwidth (sustained): 3.3 TB/s
- Footprint: <5,200 sq ft
- Power requirement: <10 MW
Trinity@LANL
Tools Challenges for Trinity

- We still have perturbation limits.... Only worse
- We still can’t collect all of the data into a huge tracefile... only worse
- Now we can’t move all of the data through the interconnect
- We have to worry about how much power our tools need ("power cap")
- Simple aggregation may hide performance problems
Current Status

Portland State University  Karen L. Karavanic  An Introduction to HPC Tools Research
MPI Tracing

- Collect MPI traces: comm type, participants
- Visualize traces with various tools, e.g. Jumpshot (ANL)

Jumpshot: Asynchronous Dynamic Load Balancing
Instruction-level Analysis

• Analyze dynamic characteristics of applications using a binary instrumentation framework, e.g., Valgrind, Pin

• Sample analyses
  – instruction mix (e.g. floating point, integer, branch, memory)
  – computational intensity (flops per memory access)
  – percentage of floating point ops that are vectorized (SIMD)
  – reuse distance
    • unique cache lines touched between two accesses to same cache line
    • gathered using detailed simulation of caches in your CPU
    • pinpoint sources of cache misses in your code
Measure and attribute costs in context
sample timer or hardware counter overflows
gather calling context using stack unwinding

Call path sample
- return address
- return address
- return address
- instruction pointer

Overhead proportional to sampling frequency...
...not call frequency
Analyzing Chombo@1024PE with hpcviewer

- Costs for:
  - inlined procedures
  - loops
  - function calls in full context
• Profiling compresses out the temporal dimension
  – temporal patterns, e.g. serialization, are invisible in profiles
• What can we do? Trace call path samples
  – sketch:
    • N times per second, take a call path sample of each thread
    • organize the samples for each thread along a time line
    • view how the execution evolves left to right
    • what do we view?
      – assign each procedure a color; view a depth slice of an execution
• Profiling compresses out the temporal dimension
  – temporal patterns, e.g. serialization, are invisible in profiles

• What can we do? Trace call path samples
  – sketch:
Load imbalance among threads appears as different lengths of colored bands along the x axis.
Formal Verification: Model Checking

Example tool: Spin (http://spinroot.com)

- Formal verification of parallel software
  - performs on-the-fly exploration of execution state spaces
- Used to identify logical design errors in parallel programs
  - e.g., communication and synchronization protocols, data structures
- Supports multiple communication models
  - message passing: both rendezvous and buffered
  - communication through shared memory
- Checks logical consistency of a specification
  - reports deadlocks, race conditions, incompleteness
  - identifies assumptions about relative speeds of processes
- Verifies properties specified with linear temporal logic
- Specify system descriptions in PROMELA modeling language
Dynamic Analysis

• **Valgrind**: framework for dynamic analysis tools
  
  http://valgrind.org
  
  – two useful valgrind tools
    
    • **memcheck**: detects memory-management problems
      
      – accesses memory it shouldn't
        
        • areas not yet allocated, areas that have been freed, areas past the end of heap blocks, inaccessible areas of the stack
      
      – reads uninitialized values
      
      – leaks memory
      
      – performs double or mismatched frees of heap blocks
    
    • **helgrind**: finds data races in multithreaded programs
      
      – memory locations accessed by >1 thread, unprotected by a lock
  
  • **A notable race detector**: cilkscreen

  Nicholas Nethercote and Julian Seward.  

Detecting data races in Cilk programs that use locks.  
A Sophisticated Shared-Memory Synchronization Algorithm

- High performance locking protocol for NUMA machines
  - lock-holder typically passes lock to a “nearby” lock-waiter
    - improves locality of shared data
  - limit lock passing to nearby threads to avoid starvation
- Accesses shared variables with RMW operations, loads, stores

Features of the Algorithm

- Tree-structured organization of queuing locks at each level
- Recursive protocol (in both acquisition and release)
- Complex interactions between participating threads at various level of hierarchy
  - Needs precise ordering of memory updates
- Generous use of
  - read-modify-write operations (compare-and-swap and swap)
  - reads and writes to shared variables
- Support for thread abort at any point in the protocol
  - protocol is “state-full”
A Concurrency Bug!

• Symptom: multiple threads simultaneously in the critical section

• Investigation
  – no obvious flaw in the algorithm
  – no obvious issue in the implementation
  – code+algorithm reviewed by multiple synchronization experts

• Traditional debugging efforts failed
  – debugging with assertions
    • assertions trigger long after the bug occurs
    • can’t backtrack to see thread interleaving that leads to assertion failure
  – single stepping in the debugger (gdb)
    • bug sporadically appears only with > 9 threads
    • does not appear when closely observed (a “Heisenbug”)
PinPlay: Deterministic Replay of Parallel Programs

1. Record a log of “buggy” execution
   (Leave overnight with multiple runs to hit assertion / segfault)
   - records access order (RAW, WAR, and WAW)
   to shared memory locations

2. Replay the execution with “buggy” thread interleaving

PinPlay: Key Features

• PinPlay records and replays
  – access order (RAW, WAR, and WAW) of shared memory locations
  – a myriad of other details needed for execution replay
  – Single-threaded, multi-threaded, and multi-process programs

• Logger: slowdown up to 147X

• Replay: slowdown up to 36X
g++ Code Generation Bug Corrupts Algorithm!

- Source code
  ```c
  // Expect atomic 64-bit write
  cacheline_aligned_64_bit_var = 0xdfffffffffffffffffd;
  ```

- GNU g++ 4.4.5 generated machine code
  ```
  movl  $0xfffffffffd,(%rax)    // write low 32-bits
  movl  $0xdfffffff,0x4(%rax) // write high 32-bits
  ```

- Splitting this 64-bit write into two parts creates a small window of inconsistent state

- Bug was not noticeable at source (the point of write)
- Bug was not noticeable at sync (read a clobbered, yet valid, 64-bit value)
- Required a record/replay tool to step through the execution under debugger to identify the inconsistent state
Interactive Debugging

- Popular tools: TotalView (Rogue Wave), DDT (Allinea)

- What can be debugged?
  - MPI applications
  - multithreaded processes
  - accelerated codes

- Laptops to supercomputers
  - debug over 100K processes

- Integrated GUI for controlling entire application
  - variable value inspection in different processes
  - data visualization

- Reverse debugging with Totalview’s Replay Engine
  - records orderings and state changes as program executes
  - recovers prior states on demand with “backward stepping”

Figure credit: http://www.allinea.com/sites/default/files/uploads/products/sparkline.png
Summary

• Lots of tools available

• Existing tools address diverse needs
  – proving parallel code correct
  – detecting data races
  – repeating intricate thread interleavings for debugging
  – interactive debugging of huge process counts
  – understanding MPI communication by tracing and visualization
  – profiling to understand where an application spends its time
  – visualizing sample traces to understand behavior over time

• Tool frameworks for building custom tools

• Challenges
  – better tools for accelerated computing
  – tools for correctness checking, debugging, and performance analysis of a billion dynamic tasks!
    • measurement, analysis, attribution, presentation
References

- Spin: http://spinroot.com
- valgrind: http://valgrind.org
- Pin: http://pintool.org
- Pinplay: http://pinplay.org
- cilkscreen: http://www.cilkplus.org
- totalview: http://www.roguewave.com
- ddt: http://www.allinea.com
- HPCToolkit: http://hpctoolkit.org
Acknowledgments

• Tools slides courtesy of John Mellor-Crummey, Department of Computer Science, Rice University

• Jumpshot figure: Rusky Lusk
Want to Learn More about Tools?

• At SC15:
  – Exhibit Floor Tools Demos:
    • IBM, The Portland Group, NVIDIA, Intel, ... also the major labs
  – Research Posters Exhibit
    • Tonight 5-7pm
  – Tech Papers
    • Wed 1:30 – 3pm Performance Tools Session

• Interested in internship/job at New Mexico Consortium?
  Contact: karavan@pdx.edu

• Interested in graduate school at Portland State University?
  Contact: karavan@pdx.edu
  – www.cs.pdx.edu/~karavan