Performance Tools for Parallel Java Environments

Sameer Shende and Allen D. Malony
Department of Computer and Information Science, University of Oregon
{sameer, malony}@cs.uoregon.edu
http://www.cs.uoregon.edu/research/paracomp/tau
Overview

- Parallel Java Environments
- TAU Performance Framework
- Performance Instrumentation
  - Java Virtual Machine Profiler Interface
  - MPI Profiling Interface
  - Integration of multiple interfaces
- An Example
- Performance Measurement Overhead
- Conclusions
- Demonstration
Parallel Java Environments

- Java applications use MPI interface for IPC
- Implementation of mpiJava
  [http://www.npac.syr.edu/projects/pcrc/HPJava/mpiJava.html]
  - JNI
  - C-Wrapper for MPI
  - One-to-One mapping Java<>C MPI Interface
- Execution of Java bytecode
  - Classic bytecode interpreter
  - JIT Compiler
  - HotSpot JVM
- Performance Analysis of Multi-Language Applications
TAU Performance Framework

- Tuning and Analysis Utilities (TAU)
- Execution Model: HPC++ Model of Computation
  - Node
  - Context
  - Thread
- Multi-level instrumentation facility
- Modular measurement framework
- Multiple thread models
- Access to CPU Performance Counters (PAPI, PCL)
- Configurable performance profiling and tracing toolkit
Architecture of TAU

Stage I: Instrumentation

Stage II: Run-Time Data Generation

Stage III: Performance Analysis

Alternative Instrumentation Modes

Run-Time Measurement

Visualization

Run-Time Library Modules

Profiling Data Files

Profile Groups

Function Database

Statistics

Event Traces

Event Table

Function Callstack

Hardware Counters

User-level Timers

TAU API

Profile

Trace

Source Code

Pre-processor

Instrumented Source Code

Compiler

Object Code

Linker

Executable Code

Operating System

Wrapper Libraries

Virtual Machine

Run to Generate Data

pprof

ASCII Report

Post-process: Merge & Convert

Vampir

pprof
Performance Instrumentation

- Multi-language applications (Java, C++, C, Fortran)
- Hybrid execution models (Java threads, MPI)
- JNI/native Java implementations of MPI Java Interface
  - Java Virtual Machine Profiler Interface (JVMPI)
  - Java Native Interface (JNI)
  - MPI Profiling Interlace
Java Virtual Machine Profiler Interface (JVMPI)

- Profiling Hooks into the Virtual Machine
- In-process profiling agent instruments Java application
- No changes to the Java source code, bytecode, or the executable code of the JVM
- Two-way call interface
- Profiler agent is a shared object (libTAU.so) loaded at runtime
- Agent registers events to the JVMPI
- JVMPI notifies events to the agent at runtime
- Agent uses JNI to invoke JVMPI control routines (mutual exclusion, etc.)
JVMPI Events

- Method transition events triggered at method entry and exits
- Memory events triggered when an object is allocated, moved, or deleted
- Heap arena events triggered when an arena is created or destroyed
- Garbage collection start and finish events
- Loading and unloading in memory events for classes and compiled methods
- JNI global and weak global reference allocation and deallocation events
- Monitor events for contended Java and raw monitors triggered when a thread attempts to enter, actually enters, or exits a monitor that is accessed by more than one thread
- Monitor wait events triggered when a thread is about to wait or finishes waiting on an object
- Thread start and end events when a thread starts or stops executing in the virtual machine
- Events that request a dump or resetting of the profiling data gathered by the in-process profiling agent
- Virtual machine initialization and shutdown events
Agent JVMPI interaction

- create a daemon thread in the virtual machine
- enable or disable the notification of an event
- enable, disable or force a garbage collection in the virtual machine
- obtain information regarding the current method call stack trace for a given thread
- obtain the accumulated CPU time consumed by the current thread
- obtain information about the object where a method took place
- get or set a pointer-sized thread-local storage data structure that can be used to record per-thread profiling data
- create or destroy a raw monitor. Raw monitors are not associated with Java objects and can be used by the profiler agent to maintain consistency of multi-threaded profiling data
- enter, exit or wait on a raw monitor for mutual exclusion. It can also notify all threads that are waiting on a raw monitor or specify a time-out period while waiting
- resume or suspend a thread
- exit the virtual machine
Integration of Multi-Level Instrumentation APIs

- Common TAU database for multiple sources
Example: Game of Life

- Profiling
  
  % prunjava 4 Life
  % racy
Example

- Tracing: Visualization in Vampir [http://www.pallas.de]

  % prunjava 4 Life
  % tau_merge tautrace*.trc Life.trc
  % tau_convert -vampir Life.trc tau.edf
  Life.pv
  % vampir Life.pv
Dynamic Call-tree Visualization

- Vampir displays TAU traces
### Performance Measurement Overhead

- TAU instrumentation overhead in microseconds

<table>
<thead>
<tr>
<th>Operation</th>
<th>Method Loading</th>
<th>Method Entry</th>
<th>Method Exit</th>
<th>Mean Overhead (μsec)</th>
<th>Standard Deviation</th>
<th>Samples</th>
<th>Range (μsec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>profiling</td>
<td>profiling</td>
<td>profiling</td>
<td>profiling</td>
<td>30.28</td>
<td>7.12</td>
<td>123</td>
<td>20.14 - 70.14</td>
</tr>
<tr>
<td>profiling</td>
<td>profiling &amp; tracing</td>
<td>profiling &amp; tracing</td>
<td>33.76</td>
<td>9.01</td>
<td>123</td>
<td>21.81 - 93.14</td>
<td></td>
</tr>
<tr>
<td>profiling</td>
<td>profiling</td>
<td>profiling</td>
<td>profiling</td>
<td>2.67</td>
<td>2.01</td>
<td>12860</td>
<td>1.14 - 50.14</td>
</tr>
<tr>
<td>profiling</td>
<td>profiling &amp; tracing</td>
<td>profiling &amp; tracing</td>
<td>4.71</td>
<td>2.82</td>
<td>12860</td>
<td>3.14 - 190.14</td>
<td></td>
</tr>
<tr>
<td>profiling</td>
<td>profiling</td>
<td>profiling</td>
<td>profiling</td>
<td>1.16</td>
<td>0.31</td>
<td>12860</td>
<td>0.14 - 15.14</td>
</tr>
<tr>
<td>profiling</td>
<td>profiling &amp; tracing</td>
<td>profiling &amp; tracing</td>
<td>2.85</td>
<td>1.29</td>
<td>12860</td>
<td>2.14 - 25.14</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- Complex parallel and distributed computing environment
- Need for observing performance events
- Requirements for an integrated portable performance analysis environment
- Constraints imposed by the system
- Unified JVM vs. native execution performance measurement
- TAU manages a multi-level, multi-threaded performance instrumentation framework
- Integrates performance instrumentation
- TAU is available from:
  - http://www.cs.uoregon.edu/research/paracomp/tau
  - http://www.acl.lanl.gov/tau