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

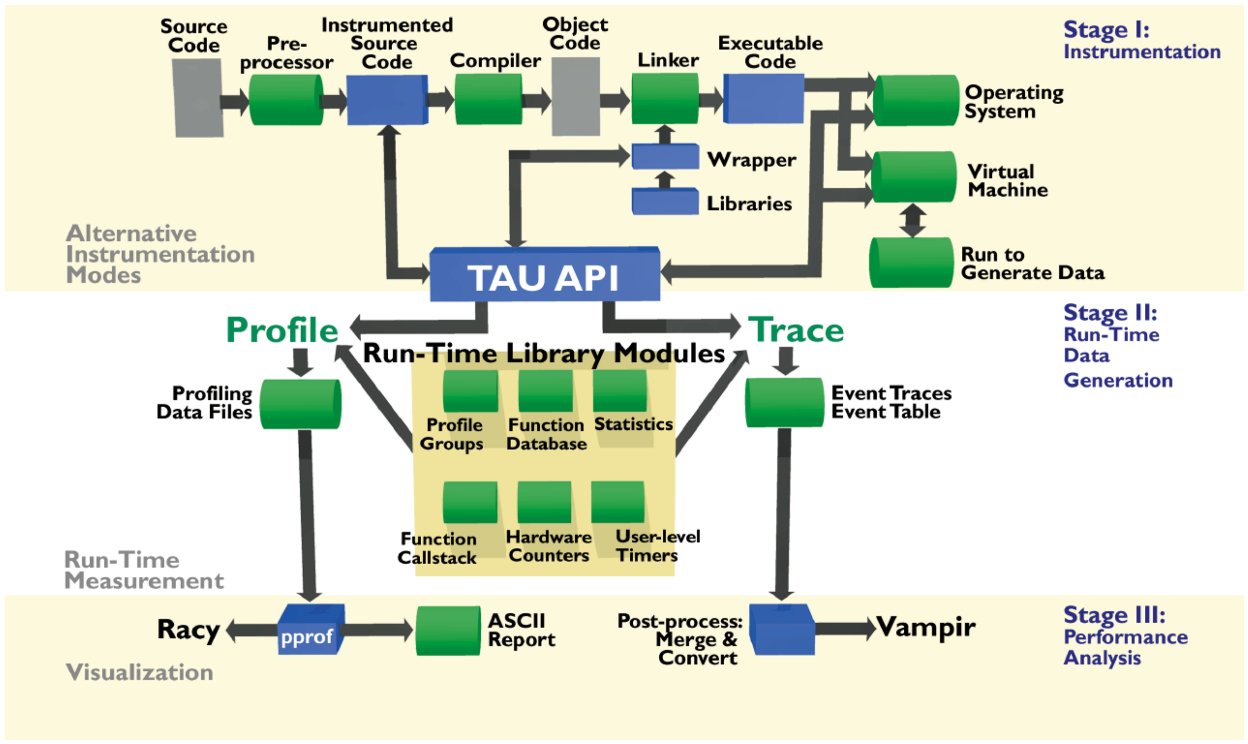
- Java applications use MPI interface for IPC
- Implementation of mpiJava
  [http://www.npac.syr.edu/projects/pcrc/HPJava/mpiJava.html]
  - JNI
  - C-Wrapper for MPI
  - One-to-One mapping Java<->C MPI Interface
- Execution of Java bytecode
  - Classic bytecode interpreter
  - JIT Compiler
  - HotSpot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

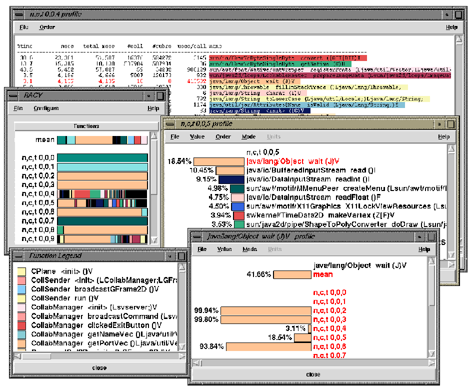
TAU Performance Framework

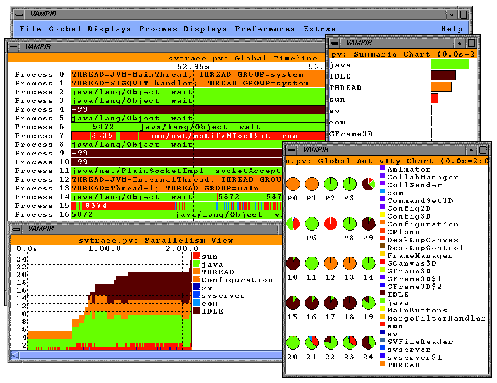
- Tuning and Analysis Utilities (TAU)
- Execution Model: HPC++ Model of Computation
  - Node
  - Context
  - Thread
- Multi-level instrumentation facility
- Modular measurement framework
- Multiple thread models
- Access to CPU Performance Counters (PAPI, PCL)
- Configurable performance profiling and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

Architecture of TAU

Tuning and Analysis Utilities







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.)

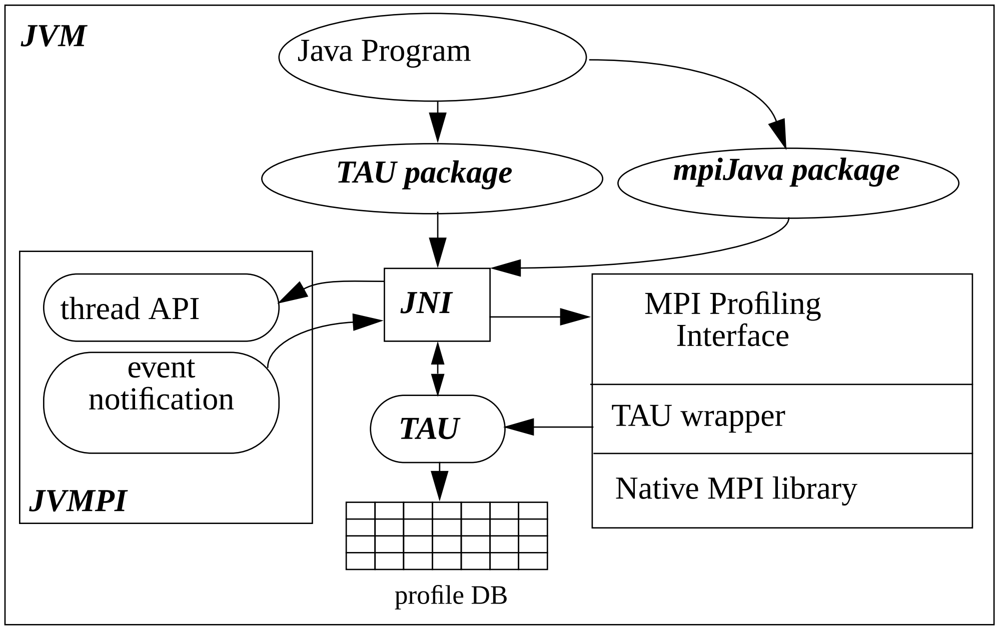
JVMPI Events

- 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



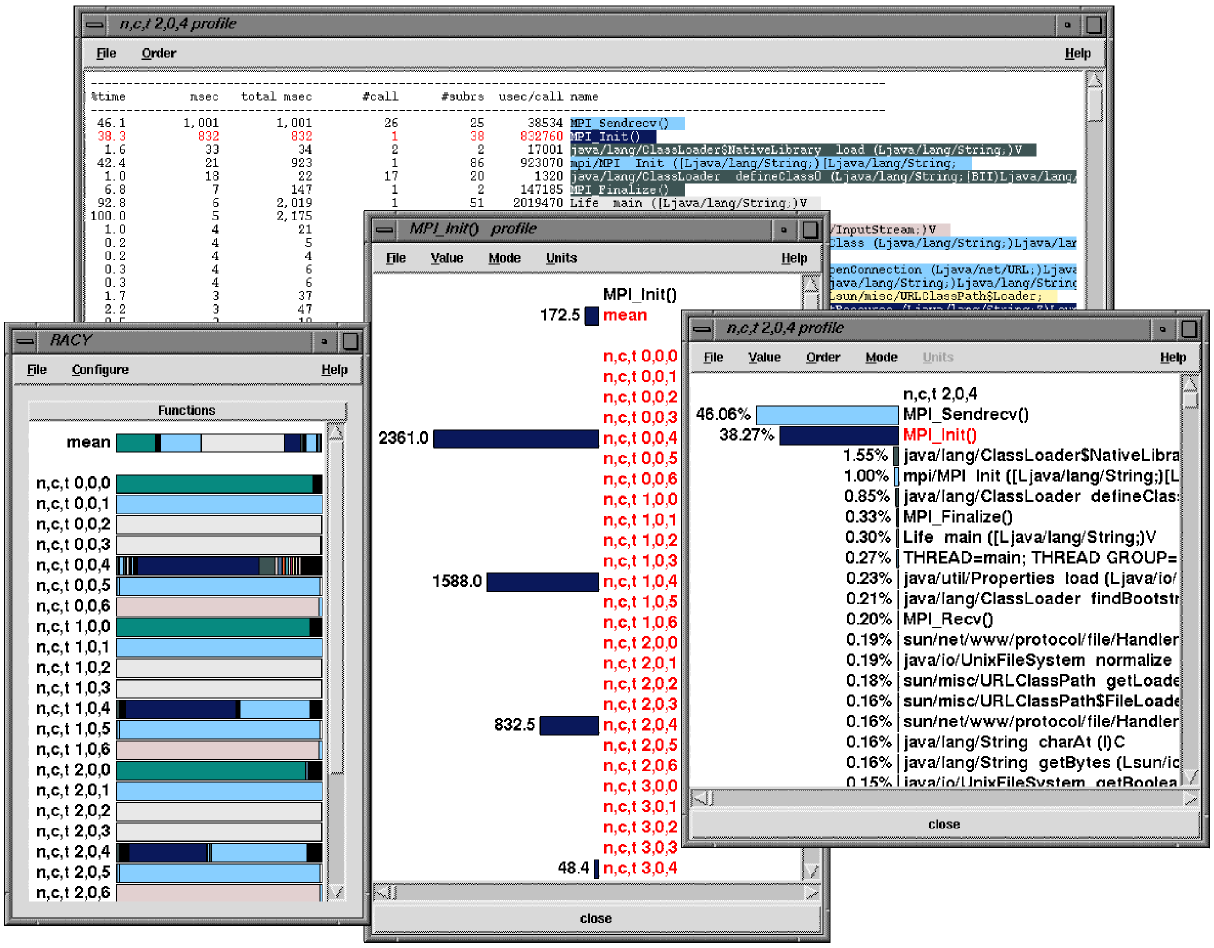
- ❑ Common TAU database for multiple sources

Example: Game of Life

- Profiling

```
% prunjava 4 Life

% racy
```



n,c,t 2,0,4 profile

```
----------------------------------------------------------------------------------------
%time          msec   total msec       #call       #subrs  usec/call name
----------------------------------------------------------------------------------------
 46.1         1,001        1,001          26          25      38534 MPI Sendrecv()
 38.3           832          832           1          38     832760 MPI_Init()
  1.6            33           34           2           2      17001 java/lang/ClassLoader$NativeLibrary  load (Ljava/lang/String;)V
 42.4            21          923           1          86     923070 mpi/MPI  Init ([Ljava/lang/String;)[Ljava/lang/String;
  1.0            18           22          17          20       1320 java/lang/ClassLoader  defineClass0 (Ljava/lang/String;[BII)Ljava/lang/
  6.8             7          147           1           2     147185 MPI Finalize()
 92.8             6        2,019           1          51    2019470 Life  main ([Ljava/lang/String;)V
100.0             5        2,175
  1.0             4           21                                    /InputStream;)V
  0.2             4            5                                    Class (Ljava/lang/String;)Ljava/lar
  0.2             4            4
  0.3             4            6                                    penConnection (Ljava/net/URL;)Ljava
  0.3             4            6                                    java/lang/String;)Ljava/lang/String
  1.7             3           37                                    Lsun/misc/URLClassPath$Loader;
  2.2             3           47
```

n,c,t 2,0,4

```
46.06% MPI_Sendrecv()
38.27% MPI_Init()
 1.55% java/lang/ClassLoader$NativeLibra
 1.00% mpi/MPI  Init ([Ljava/lang/String;)[L
 0.85% java/lang/ClassLoader  defineClas
 0.33% MPI_Finalize()
 0.30% Life  main ([Ljava/lang/String;)V
 0.27% THREAD=main; THREAD GROUP=
 0.23% java/util/Properties  load (Ljava/io/
 0.21% java/lang/ClassLoader  findBootstr
 0.20% MPI_Recv()
 0.19% sun/net/www/protocol/file/Handler
 0.19% java/io/UnixFileSystem  normalize
 0.18% sun/misc/URLClassPath  getLoade
 0.16% sun/misc/URLClassPath$FileLoade
 0.16% sun/net/www/protocol/file/Handler
 0.16% java/lang/String  charAt (I)C
 0.16% java/lang/String  getBytes (Lsun/ic
 0.15% java/io/UnixFileSystem  getBoolea
```

Example

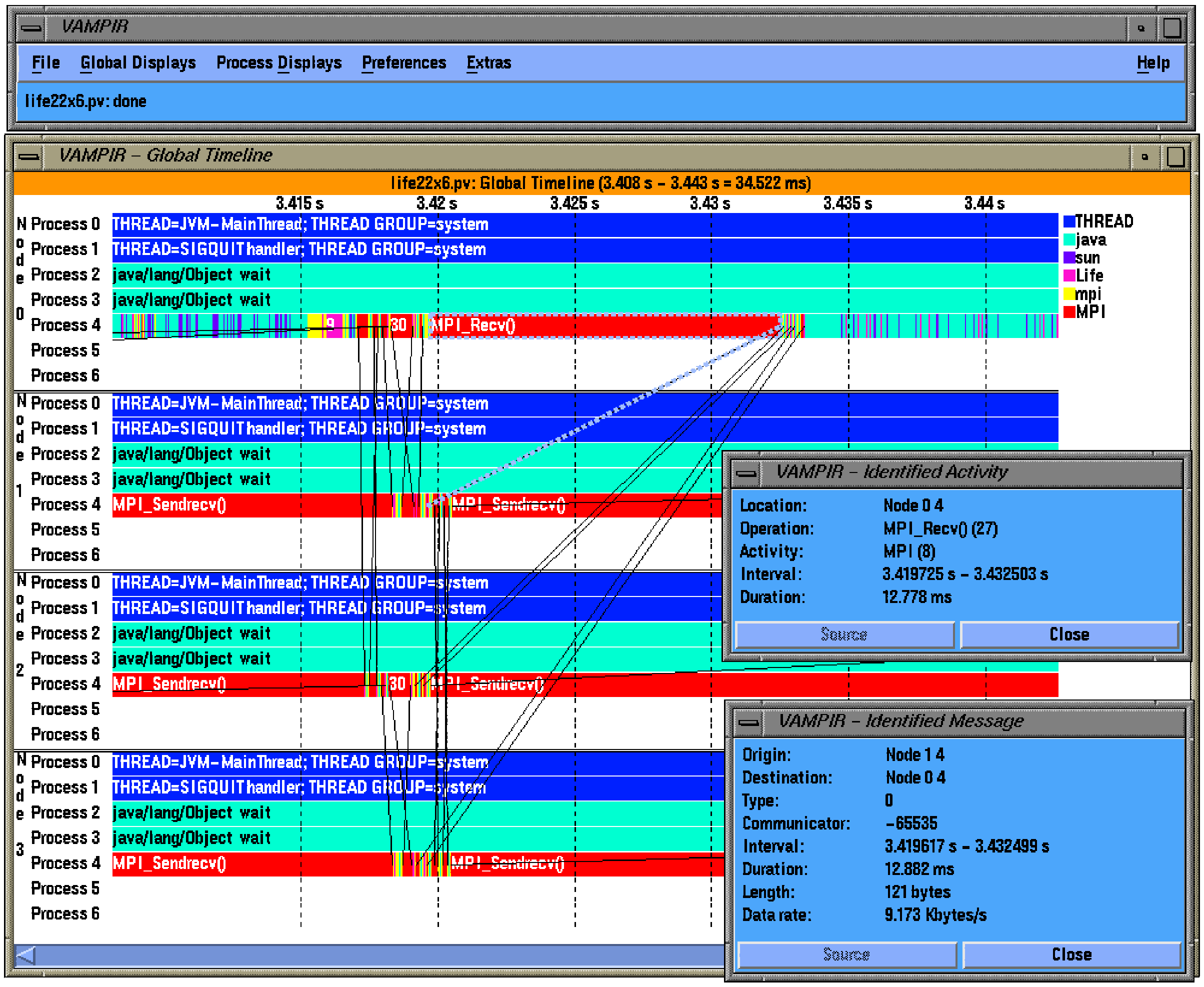
❑ Tracing: Visualization in Vampir [http://www.pallas.de]

```
% prunjava 4 Life

% tau_merge tautrace*.trc Life.trc

% tau_convert -vampir Life.trc tau.edf
Life.pv

% vampir Life.pv
```



life22x6.pv: done

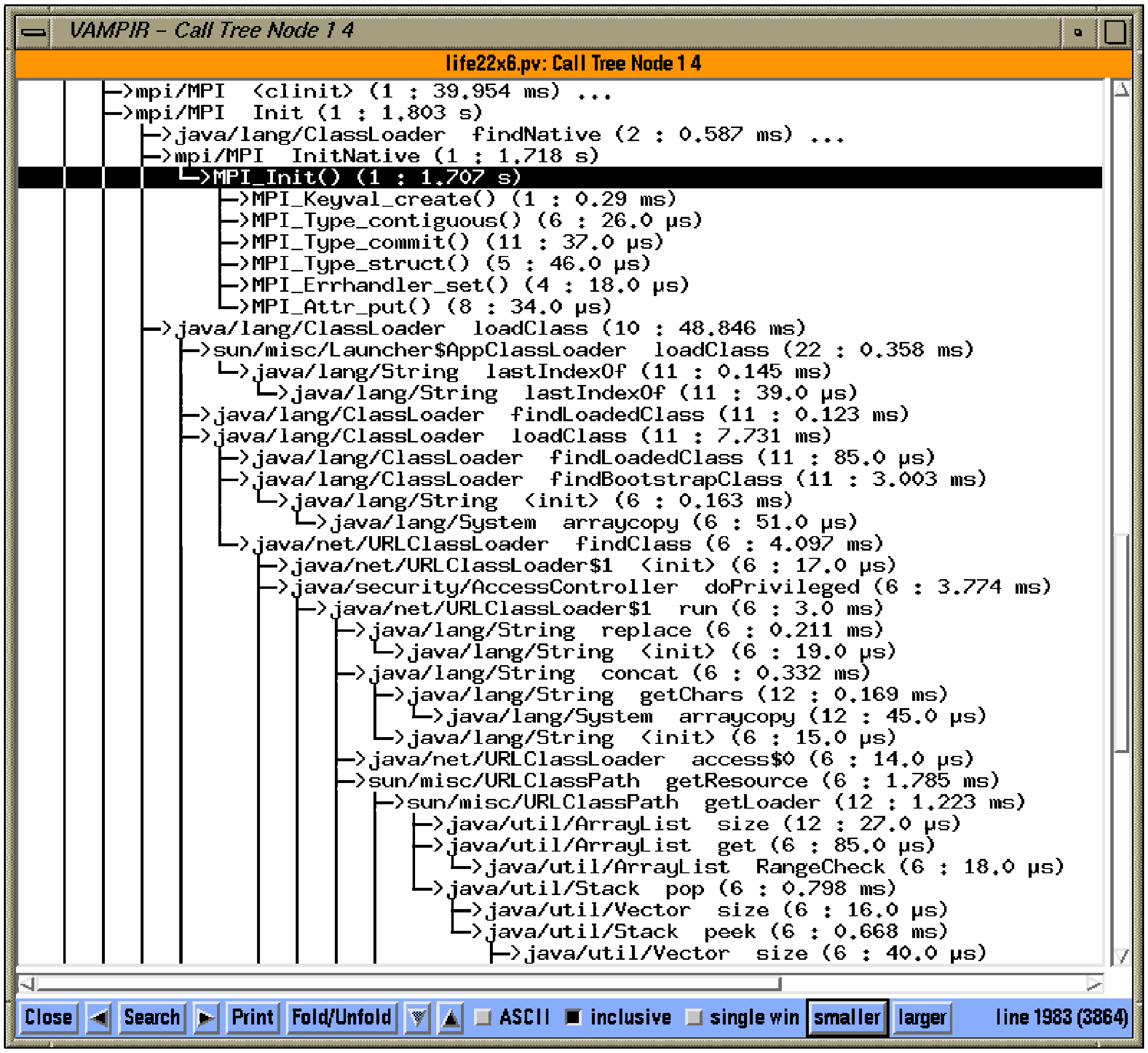
life22x6.pv: Global Timeline (3.408 s – 3.443 s = 34.522 ms)

Location:	Node 0 4
Operation:	MPI_Recv() (27)
Activity:	MPI (8)
Interval:	3.419725 s – 3.432503 s
Duration:	12.778 ms

Origin:	Node 1 4
Destination:	Node 0 4
Type:	0
Communicator:	-65535
Interval:	3.419617 s – 3.432499 s
Duration:	12.882 ms
Length:	121 bytes
Data rate:	9.173 Kbytes/s

Dynamic Call-tree Visualization

- Vampir displays TAU traces



```
├─>mpi/MPI  <clinit> (1 : 39.954 ms) ...
├─>mpi/MPI  Init (1 : 1.803 s)
│  ├─>java/lang/ClassLoader  findNative (2 : 0.587 ms) ...
│  ├─>mpi/MPI  InitNative (1 : 1.718 s)
│  │  └─>MPI_Init() (1 : 1.707 s)
│  │     ├─>MPI_Keyval_create() (1 : 0.29 ms)
│  │     ├─>MPI_Type_contiguous() (6 : 26.0 µs)
│  │     ├─>MPI_Type_commit() (11 : 37.0 µs)
│  │     ├─>MPI_Type_struct() (5 : 46.0 µs)
│  │     ├─>MPI_Errhandler_set() (4 : 18.0 µs)
│  │     └─>MPI_Attr_put() (8 : 34.0 µs)
│  ├─>java/lang/ClassLoader  loadClass (10 : 48.846 ms)
│  │  ├─>sun/misc/Launcher$AppClassLoader  loadClass (22 : 0.358 ms)
│  │  │  └─>java/lang/String  lastIndexOf (11 : 0.145 ms)
│  │  │     └─>java/lang/String  lastIndexOf (11 : 39.0 µs)
│  │  ├─>java/lang/ClassLoader  findLoadedClass (11 : 0.123 ms)
│  │  ├─>java/lang/ClassLoader  loadClass (11 : 7.731 ms)
│  │  │  ├─>java/lang/ClassLoader  findLoadedClass (11 : 85.0 µs)
│  │  │  ├─>java/lang/ClassLoader  findBootstrapClass (11 : 3.003 ms)
│  │  │  │  └─>java/lang/String  <init> (6 : 0.163 ms)
│  │  │  │     └─>java/lang/System  arraycopy (6 : 51.0 µs)
│  │  │  └─>java/net/URLClassLoader  findClass (6 : 4.097 ms)
│  │  │     ├─>java/net/URLClassLoader$1  <init> (6 : 17.0 µs)
│  │  │     ├─>java/security/AccessController  doPrivileged (6 : 3.774 ms)
│  │  │     │  ├─>java/net/URLClassLoader$1  run (6 : 3.0 ms)
│  │  │     │  │  ├─>java/lang/String  replace (6 : 0.211 ms)
│  │  │     │  │  │  └─>java/lang/String  <init> (6 : 19.0 µs)
│  │  │     │  │  ├─>java/lang/String  concat (6 : 0.332 ms)
│  │  │     │  │  │  ├─>java/lang/String  getChars (12 : 0.169 ms)
│  │  │     │  │  │  │  └─>java/lang/System  arraycopy (12 : 45.0 µs)
│  │  │     │  │  │  └─>java/lang/String  <init> (6 : 15.0 µs)
│  │  │     │  │  ├─>java/net/URLClassLoader  access$0 (6 : 14.0 µs)
│  │  │     │  │  ├─>sun/misc/URLClassPath  getResource (6 : 1.785 ms)
│  │  │     │  │  │  ├─>sun/misc/URLClassPath  getLoader (12 : 1.223 ms)
│  │  │     │  │  │  │  ├─>java/util/ArrayList  size (12 : 27.0 µs)
│  │  │     │  │  │  │  ├─>java/util/ArrayList  get (6 : 85.0 µs)
│  │  │     │  │  │  │  │  └─>java/util/ArrayList  RangeCheck (6 : 18.0 µs)
│  │  │     │  │  │  │  └─>java/util/Stack  pop (6 : 0.798 ms)
│  │  │     │  │  │  │     ├─>java/util/Vector  size (6 : 16.0 µs)
│  │  │     │  │  │  │     └─>java/util/Stack  peek (6 : 0.668 ms)
│  │  │     │  │  │  │        ├─>java/util/Vector  size (6 : 40.0 µs)
```

Performance Measurement Overhead

- TAU instrumentation overhead in microseconds

Operation		Mean Overhead (μsec)	Standard Deviation	Samples	Range (μsec)
Method Loading	profiling	30.28	7.12	123	20.14 - 70.14
	profiling & tracing	33.76	9.01	123	21.81 - 93.14
Method Entry	profiling	2.67	2.01	12860	1.14 - 50.14
	profiling & tracing	4.71	2.82	12860	3.14 - 190.14
Method Exit	profiling	1.16	0.31	12860	0.14 - 15.14
	profiling & tracing	2.85	1.29	12860	2.14 - 25.14

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