# Dynamic Instrumentation of Threaded Applications

**Zhichen Xu, Bart P. Miller and Oscar Naim**

zhichen@cs.wisc.edu

Computer Science Department
University of Wisconsin
1210 W. Dayton St.
Madison, WI 53706-1685

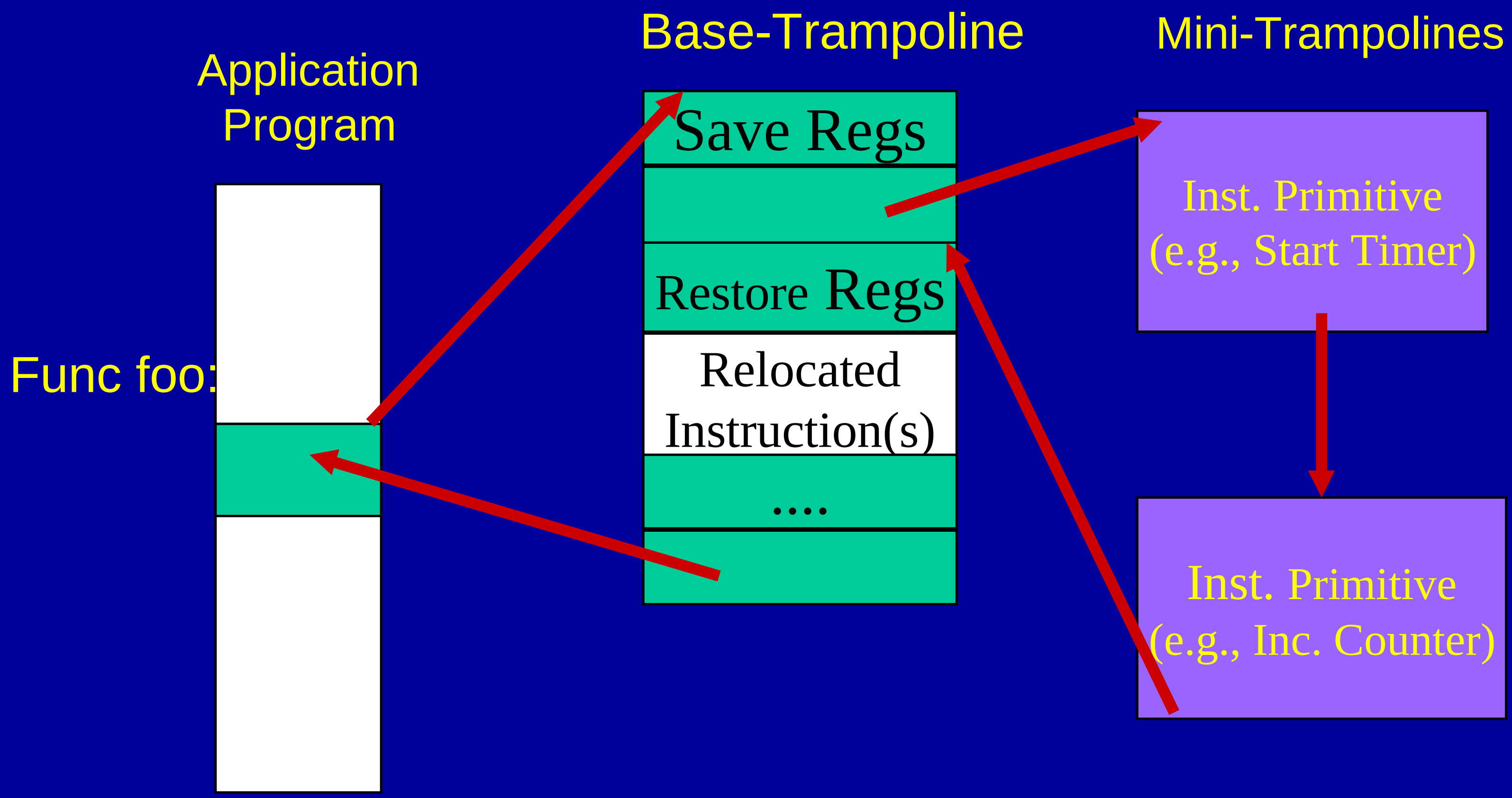
# Introduction

- Use of threads is becoming common
  - Database and web servers, Java interpreters, Internet search engines, graphical user interfaces, irregular numerical applications, etc.
  - More obstacles to good performance
- However
  - Few tools monitor threaded programs
  - Threaded programs are hard to instrument

# Instrumenting Threaded Programs

- Main Techniques
  - Same instrumentation code multiple data
  - Thread-conscious lock to avoid self-deadlock
  - Per-thread virtual timers
  - Safe inferior RPC
- Extend Paradyn to profile threads (Solaris)
- Initial experience
  - Speedup a Java native method by 42%.
  - Increase by 24% the amount of work in unit time

# Instrumentation without Threads

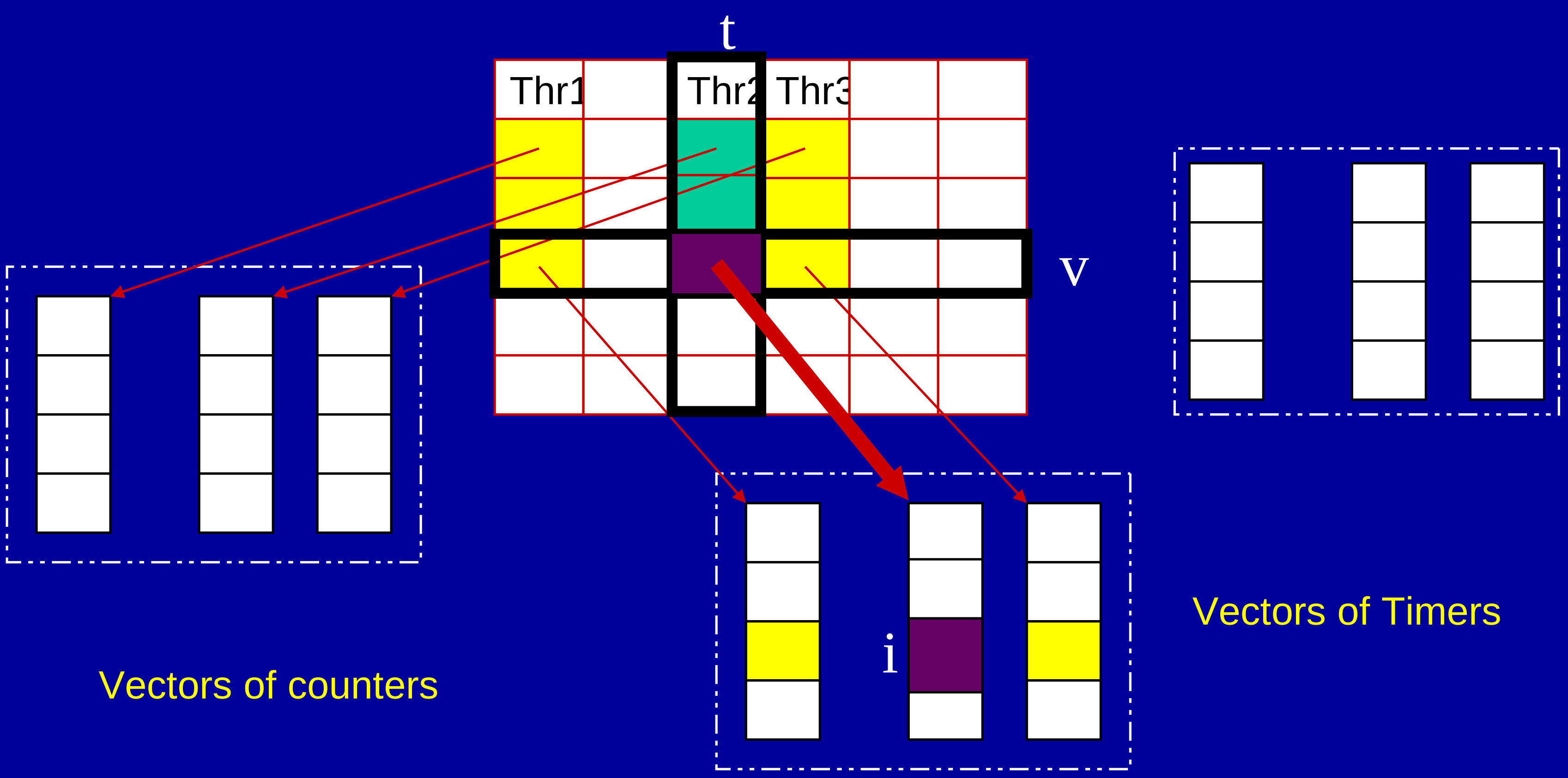


# Same Instrumentation Code Multiple Data

- All threads share instrumentation
- Each thread has private copy of counters/timers
- Always allocate counters/timers for active threads
- Instrumentation code figures out which data
- Compute cumulative metrics by aggregating measurements for individual threads

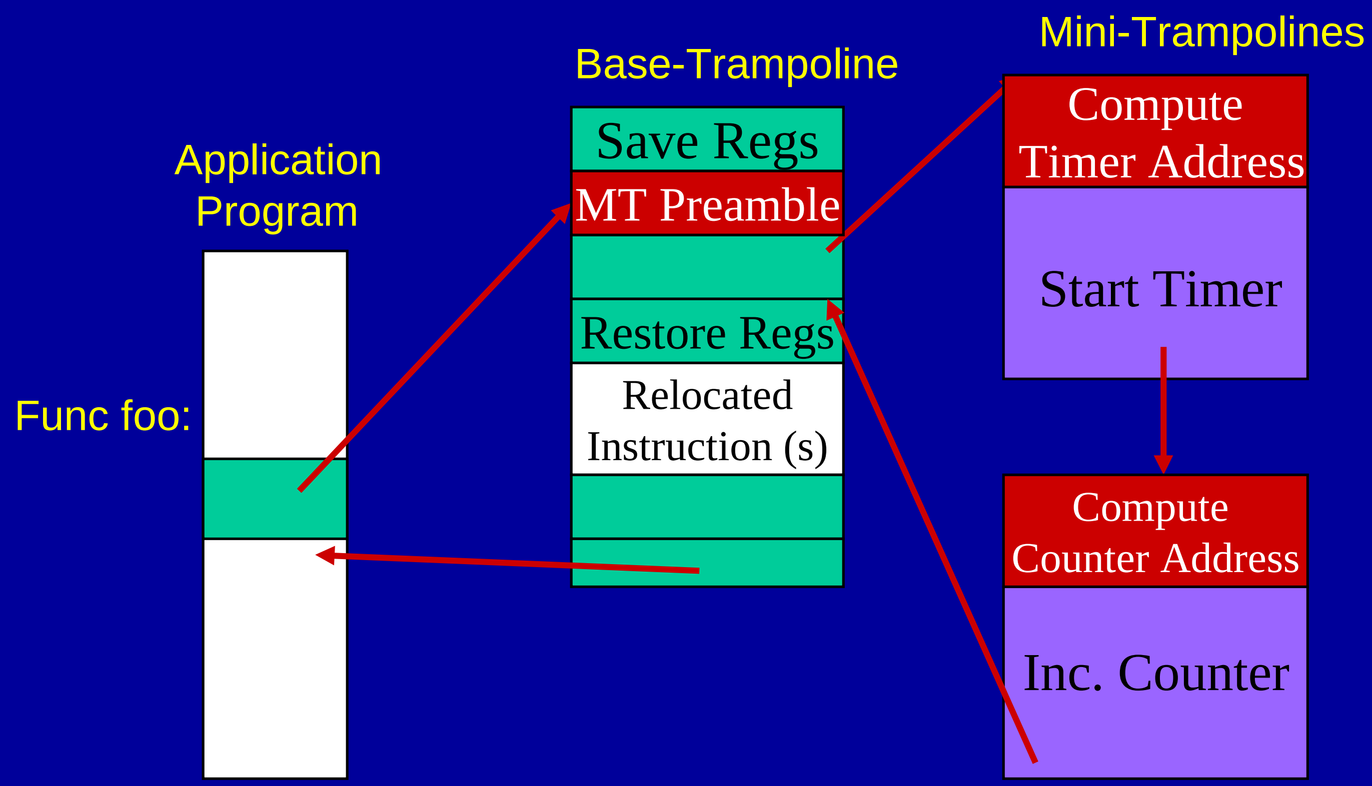
# Data Heap - Thread Table

- Each active thread is allocated a column (t)
- Counter/Timer address given by [t,(v,i)]



# Instrumentation Code

- MT Preamble returns the column index (t) of the current thread
- Mini-tramp. uses [t,(v,i)] to compute counter/timer address



# Same Instrumentation Code Multiple Data (cont'd)

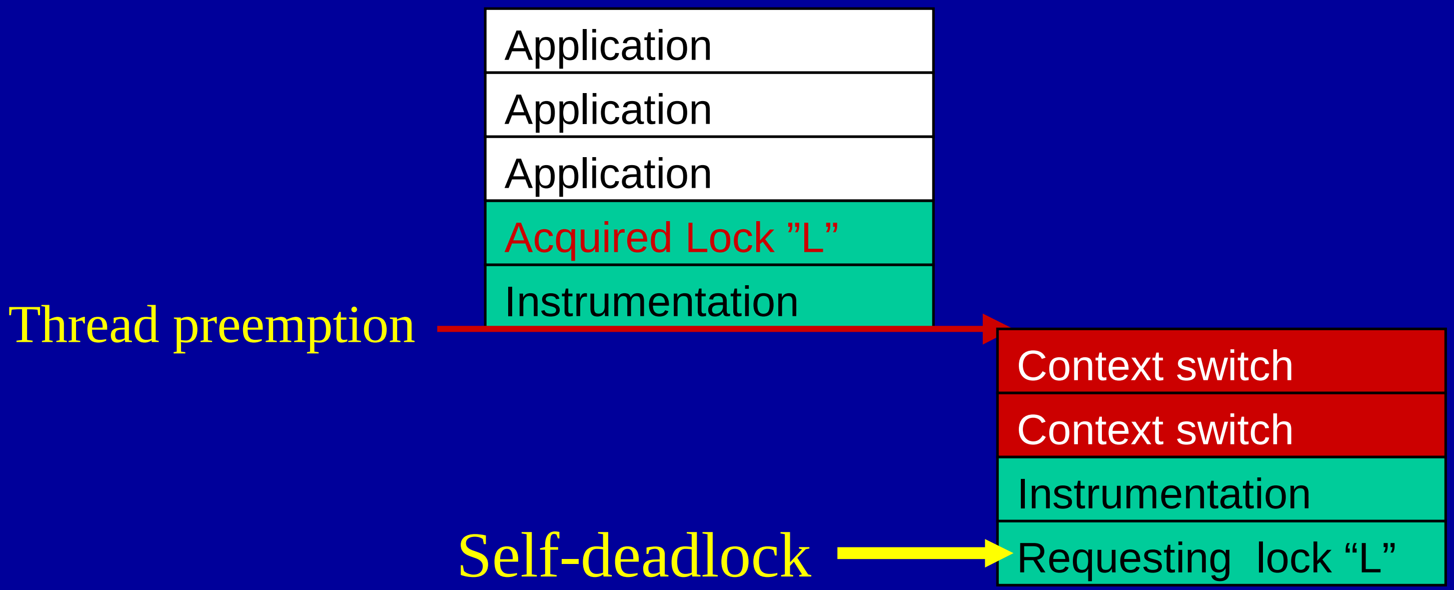
+ Trampoline similar to non-threaded version
+ No locks are needed for counters/timers
+ Address calculation is simple and efficient
- Some counter/timer may never be used

# But, It is not Really That Easy!

- We need to:
  - Use locks to guard global data structures
  - Instrument thread context switches
    - (e.g., implementing time-based metrics).
    - Could cause deadlock
  - Trigger instrumentation after execution has passed the insertion point (inferior RPC)

# Instrumenting Thread Switches

- Interleaving of instrumentation
- Self-deadlock



# Instrumenting Thread Switches (cont'd)

## Thread-Conscious Lock

| Previous | | New | | Return Value |
|---|---|---|---|---|
| State | Tid | State | Tid | |
| Held | t1 | Held | t1 | Self |
| Held | t2 | Held | t2 | No |
| Free | - | Held | t1 | Yes |

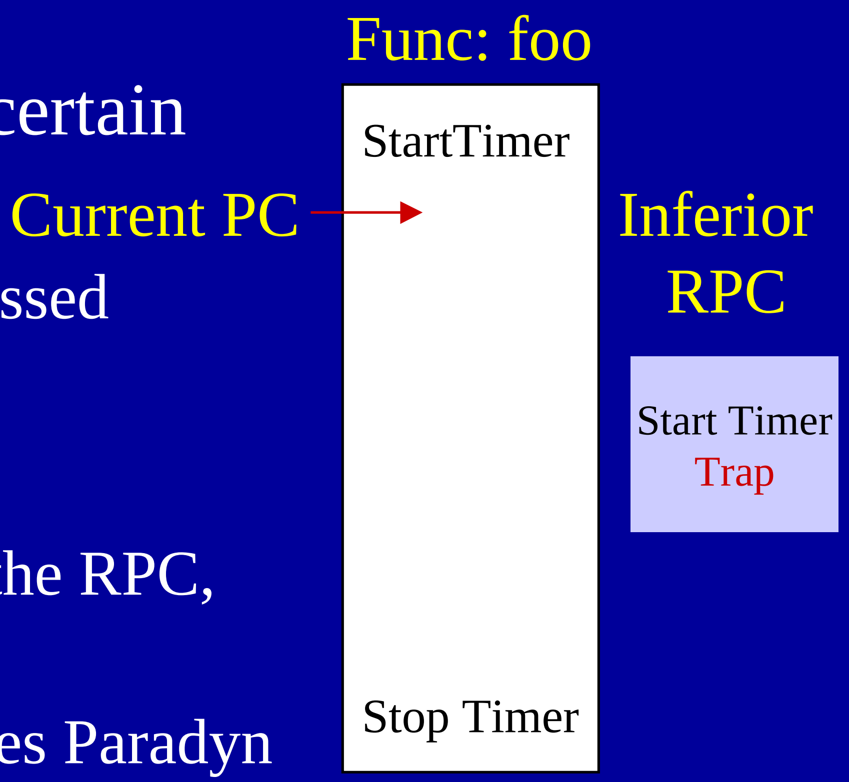
Thread t1 requests tc-lock(*l*), where *l* may already be held

# Per-thread Virtual Timers

- Problem
  – No system call to measure CPU spent in a thread
  – But can get  CPU time for a light-weighted process
- Solution
  – Use LWP timer where a thread is mapped
  – Stop/restart at thread switch out/in
  – Switch LWP timer if a thread migrates
  + reduce expensive timer calls.
  + Reduce chance of interleaving instrumentation.

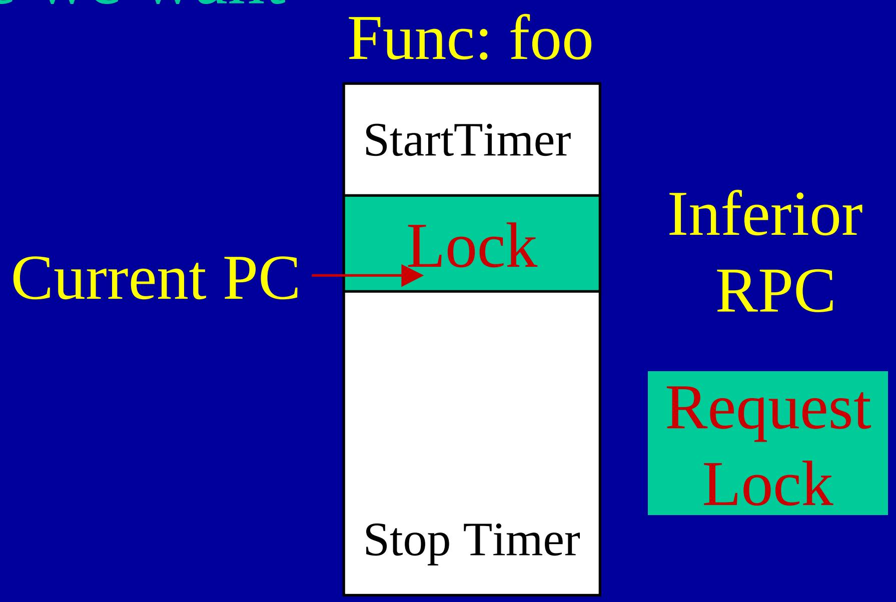
# Inferior RPC (aka oneTimeCode)

- Force application to execute certain instrumentation code
  - Needed when execution has passed insertion point
- Implementation
  - Pause the application, install the RPC, change PC to RPC code
  - A trap at the end of RPC notifies Paradyn to resume application



# Inferior RPC (cont'd)

- Problems with threads:
  - Need to execute RPC for a particular thread
  - When perform inferior RPC, the thread could be the one we want



# Inferior RPC (cont'd)

- Solution
  - Any thread can execute RPC for another thread
  - Pass an extra parameter to identify thread
  - Post RPC in shared-memory
  - Add code in every base-tramp. to check pending RPC

# Instrumentation Overhead

- Instrumentation primitives
  - Base-trampoline: 5x
  - Counter primitives: 1.5x
  - Timer primitives: 1.3x-1.4x

| Machine | Base Tramp. | | Counter | | Start Timer | | Stop Timer | |
|---|---|---|---|---|---|---|---|---|
| | Non-threaded | Threaded | Non-threaded | Threaded | Non-threaded | Threade d | Non-Threaded | Threaded |
| **UltraSPARC II Uniprocessor** | **125ns** | **552ns 5x** | **28ns** | **41ns 1.5x** | **1.1us** | **1.5us 1.4x** | **1.2us** | **1.5us 1.3x** |
| **Enterprise 5000s** | **186ns** | **815ns 5x** | **42ns** | **65ns 1.5x** | **1.5us** | **2.2us 1.4x** | **1.6us** | **2.0us 1.3x** |

# Instrumentation overhead (cont'd)

- Two versions of matrix multiply
  - Intrusion for thread instrumentation: 1x-7x

| **Instrumentation** | **Non-threaded Paradyn (Sequential Version)** | | **Threaded Paradyn (Threaded Version)** | |
|---|---|---|---|---|
| | UltraSPARC II 1 processor | Enterprise 5000s | UltraSPARC II 1 processor | Enterprise 5000s |
| **No Instrumentation** | **64.6s** | **95.9s** | **64.7s** | **24.4s** |
| **CPU Time (Inclusive) Whole program** | **65.3s (+1.1%)** | **96.3s (+0.4%)** | **65.6s (+1.4%)** | **24.5s (+0.4%)** |
| **Procedure Call Frequency (innerp)** | **65.4s (+1.2%)** | **96.4s (+0.5%)** | **66.7s (+3.1%)** | **25.4s (+4.1%)** |
| **CPU Time (Inclusive, innerp)** | **66.6s (+3.1%)** | **97.9s (+2.1%)** | **68.4s (+5.7%)** | **25.9s (+6.1%)** |

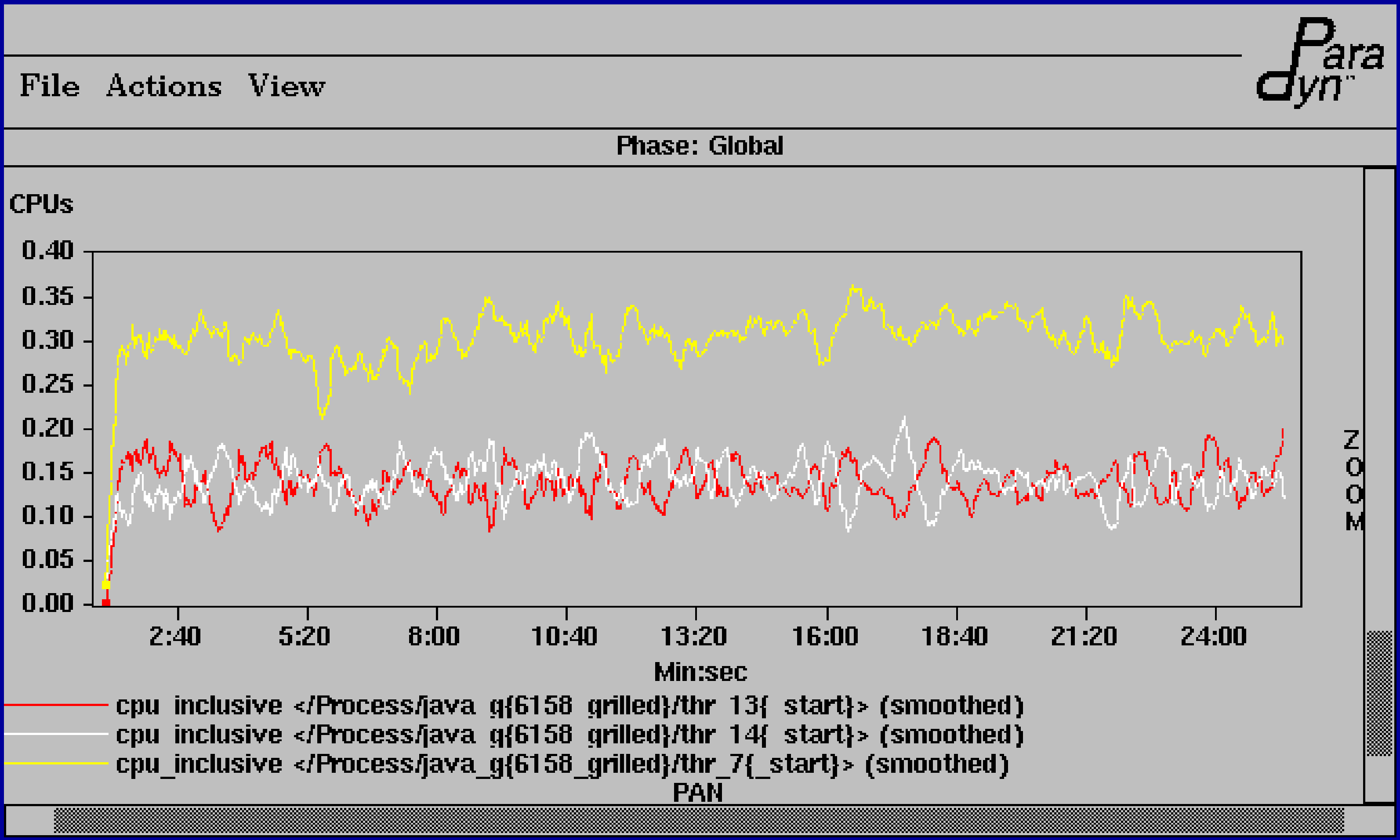
# Performance Study

- Benchmark
  - Sun Java Virtual Machine
  - Interpreting the AppletViewer
  - Driven by a game applet, Tetris (1000 lines)
- Performance improvements
  - Redundant code elimination

| | sun_awt_motif_X11Graphics_drawLine | Lines/Second |
|---|---|---|
| Original | 6.7us | 9,474 |
| Optimized | 3.9 us (-42%) | 11,718 (+24%) |

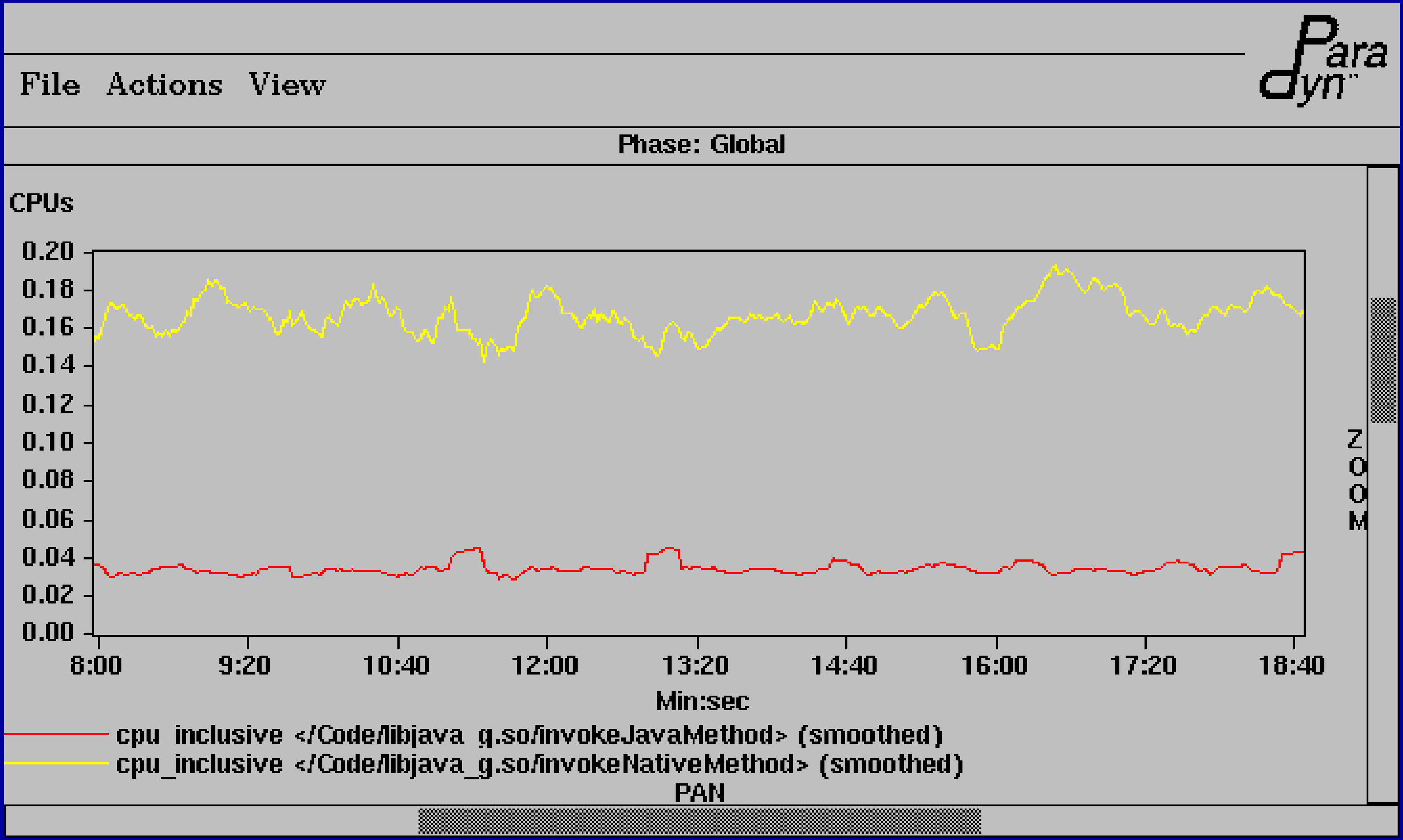
# Performance Study (cont'd)

- CPU time on a per-thread basis (thr_7, thr_13, thr_14)



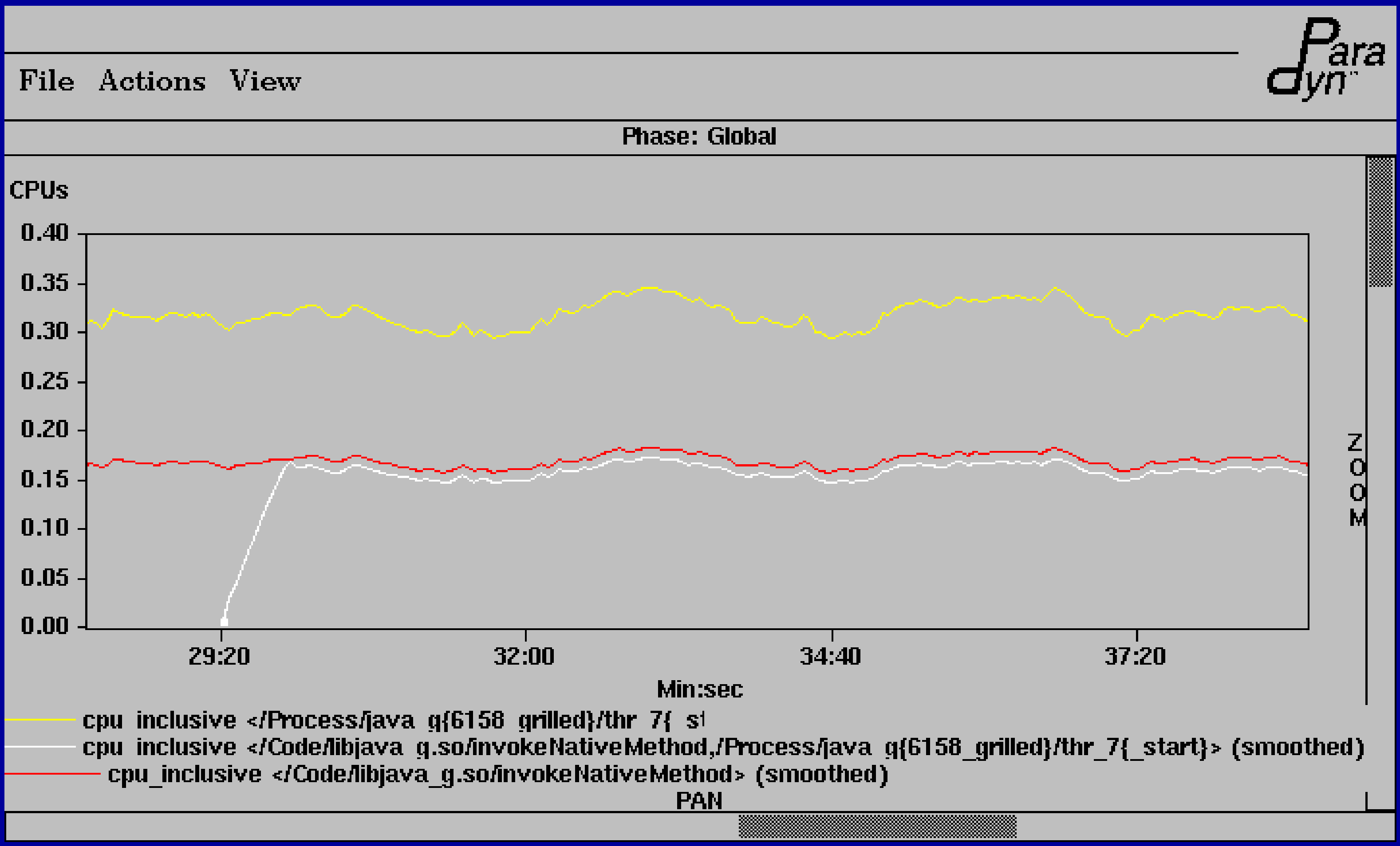
# Performance Study (cont'd)

- invokeNativeMethod takes about 20% of total CPU



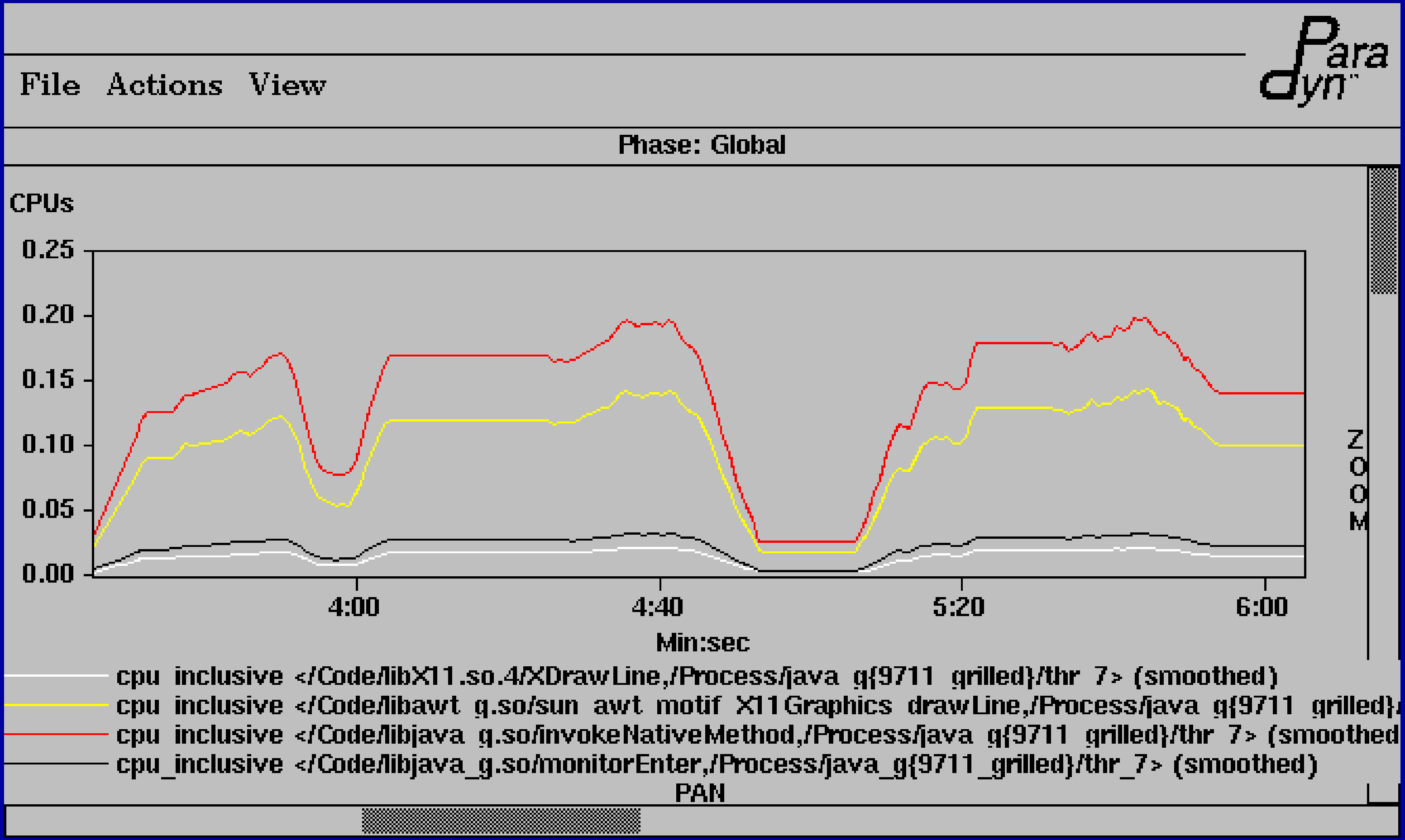
# Performance Study (cont'd)

- invokeNativeMethod mostly called by thr_7



# Performance Study (cont'd)

- Subdivide invokeNativeMethod on thr_7



# Portability Issues

- Profile both threaded and non-threaded applications with minimal overhead
  - base-Tramp. : 2x; counter: 1.5x; timer: 1x
- Support different thread packages
  - thread creation/deletion routines
  - thread context switch routines
  - Get information about active threads
    - LWP, stack, and etc.

# Conclusion

- Paradyn can now instrument threaded programs
- Instrumentation overhead reasonable
- Used to tune performance of a large threaded application (Sun JVM).
- Future work
  - Reduce instrumentation overhead.
  - More thread-specific metrics
  - Integrate with released version