Performance Measurement of Dynamically Compiled Java Executions

Tia Newhall and Barton P. Miller

{newhall*, bart}@cs.wisc.edu

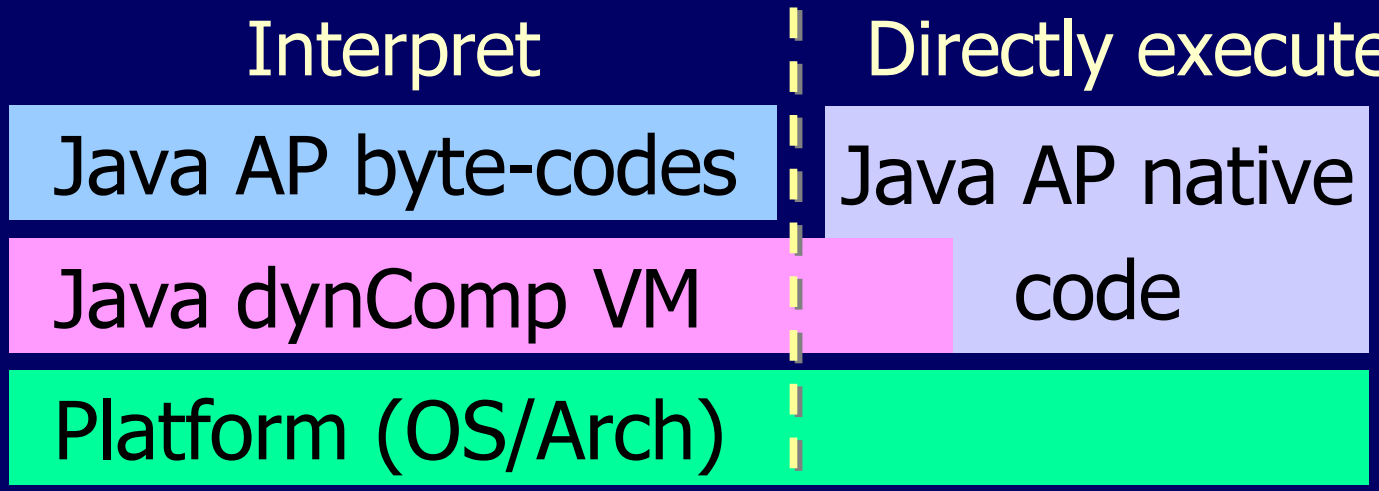
Computer Sciences
University of Wisconsin
1210 W. Dayton St.
Madison, WI 53706
http://www.cs.wisc.edu/~newhall

* Swarthmore College Computer Sciences Department

Motivation for profiling tool

- Java is increasingly being used for large, long-running, complex applications
  - Meta-computing
  - High performance numeric applications
  - Parallel computing
- Dynamic Compiler Java virtual machines will become ubiquitous
  - native code execution + run-time optimizations
  - potential to outperform statically compiled code

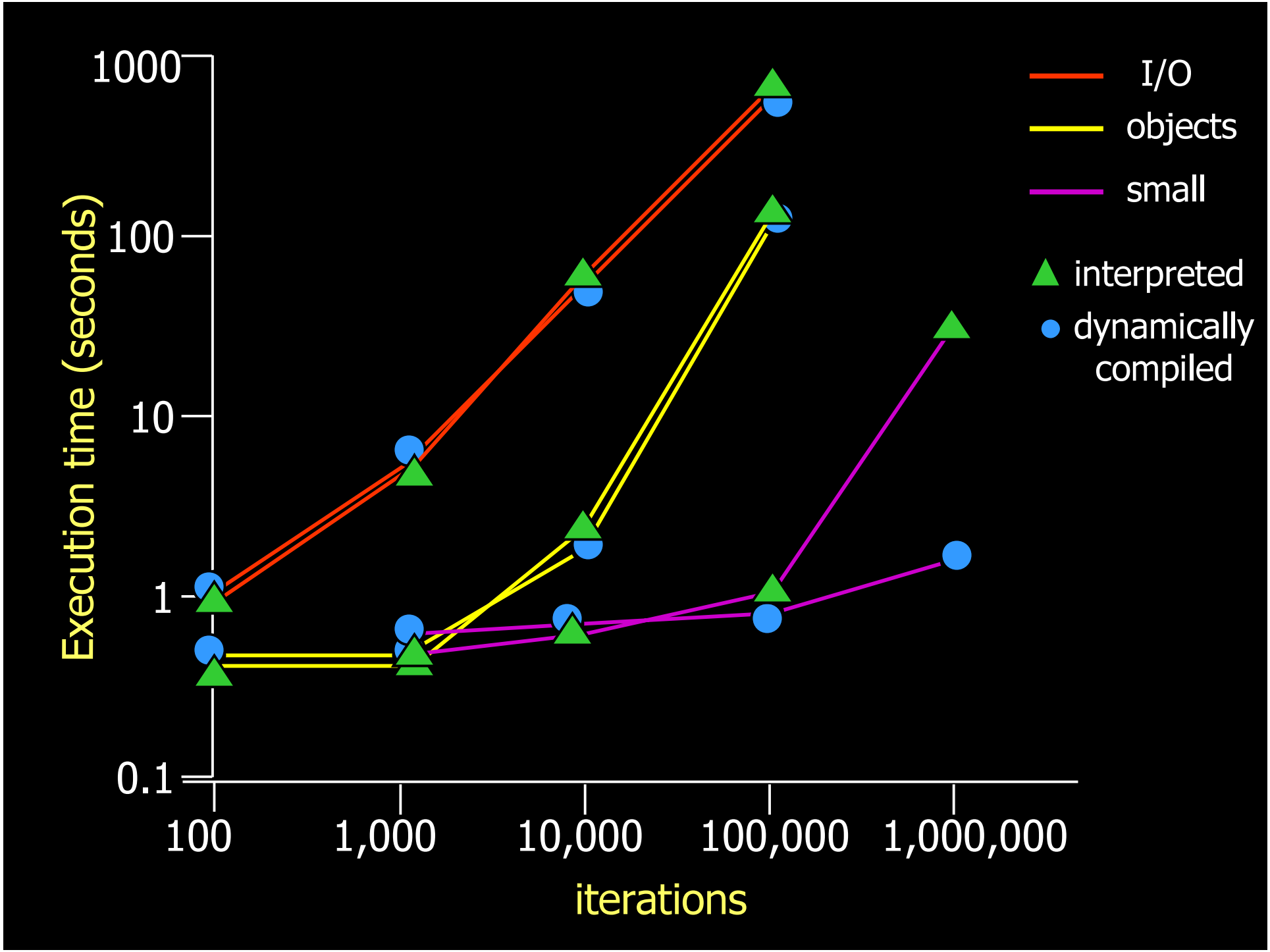
Profiling DC Java is hard



- Java application changes form at run-time
  - discover mappings to correlate data
  - find size & location of native AP to measure it
- Run-time interactions between VM and AP
  - describe VM interactions with AP native code

Performance issues of dynamically compiled Java

- When dynamically compiling doesn't win:
  - small method functions with simple CFG's
  - methods whose time not dominated by interpreting byte-code (I/O or synchronization)
  - methods whose native code form still has a lot of interaction with Java VM (object creates)
- Simple study:
  - run application kernels on ExactVM & compare all-interpreted to dynamically compiled execution


1000
100
10
1
0.1
Execution time (seconds)
100
1,000
10,000
100,000
1,000,000
iterations
I/O
objects
small
interpreted
dynamically compiled

We need a profiling tool

- Dynamic compilation is not the only answer
- Need more information to tune application
  - performance measures with native code form and byte-code form of a method
    - did run-time compilation help?
  - VM interactions with native code form of a method
    - what are these interactions?
    - how much do they affect the application's execution?

Paradyn-J

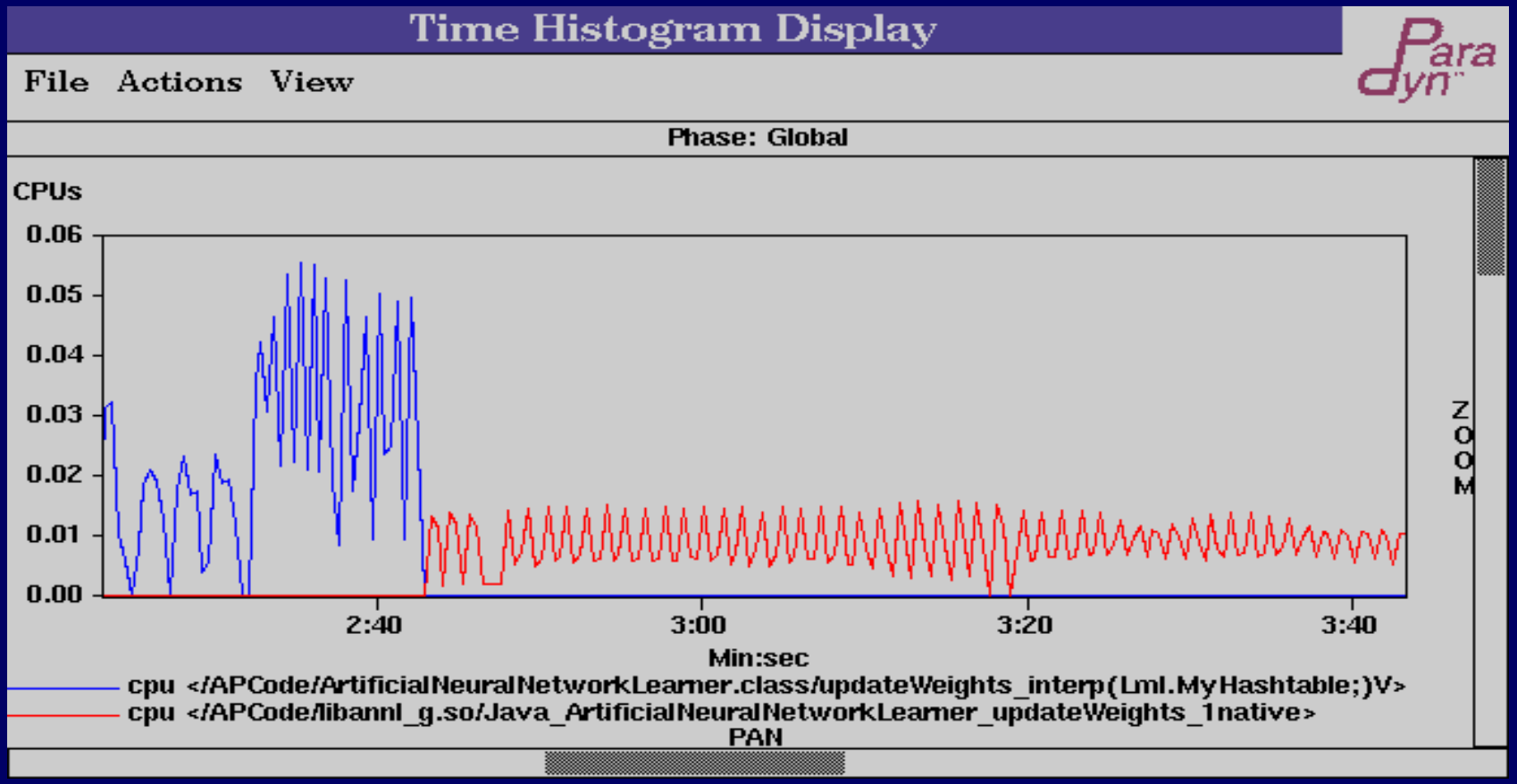
- Extension of Paradyn Parallel Performance Tools for measuring Java executions
  - profiles simulation of dynamically compiled Java
  - dynamically inserts native and byte-code instrumentation in VM & AP at run-time

  + instrument unmodified Java .class files and VM
- Provides performance data that:
  - associated with AP's multiple execution forms
  - describes VM-AP interactions (see EuroPar'98)
  - describes run-time compilation costs

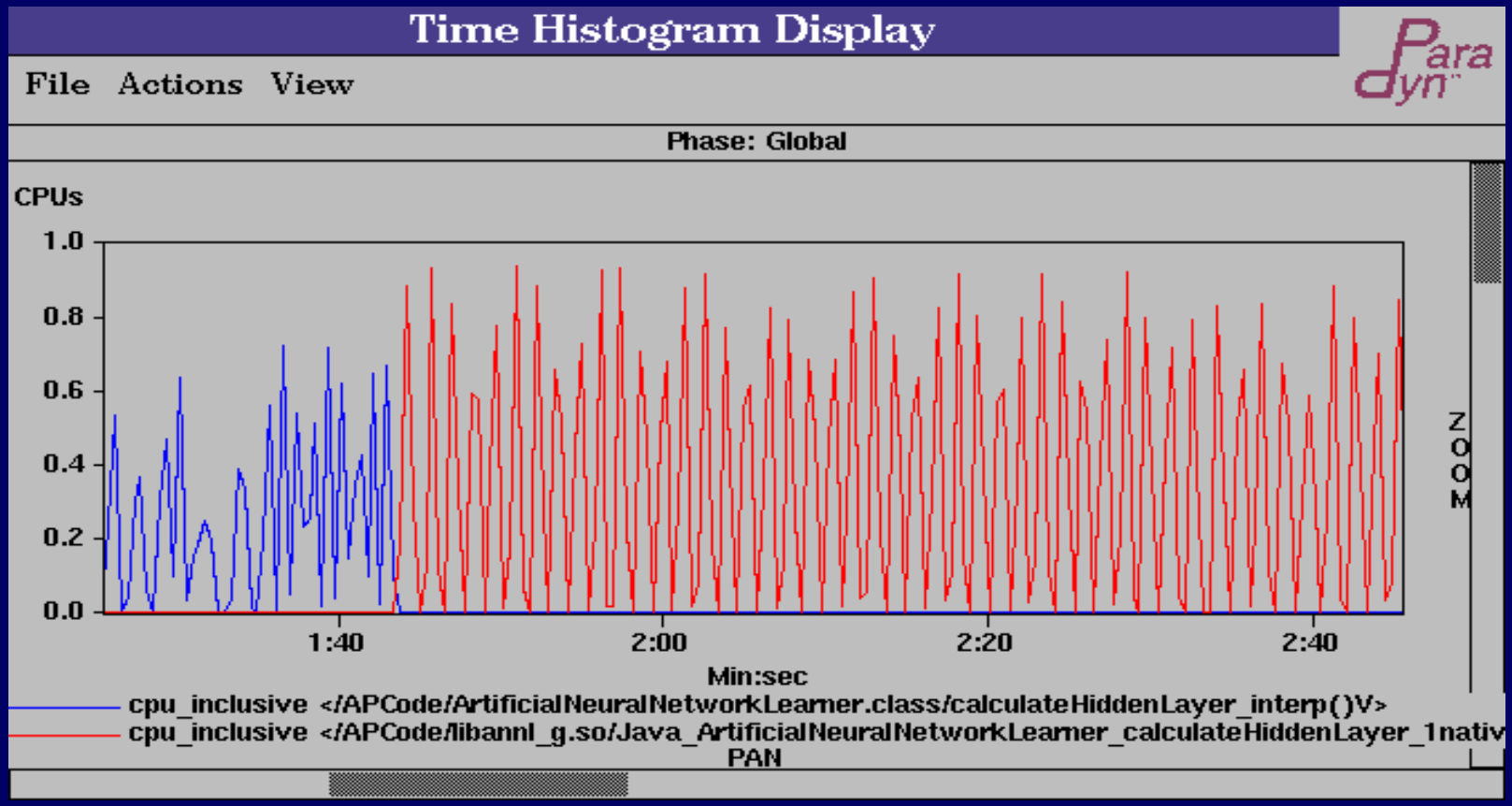
Performance tuning study

Neural network application

(15,800 lines of source code, 23 class files)

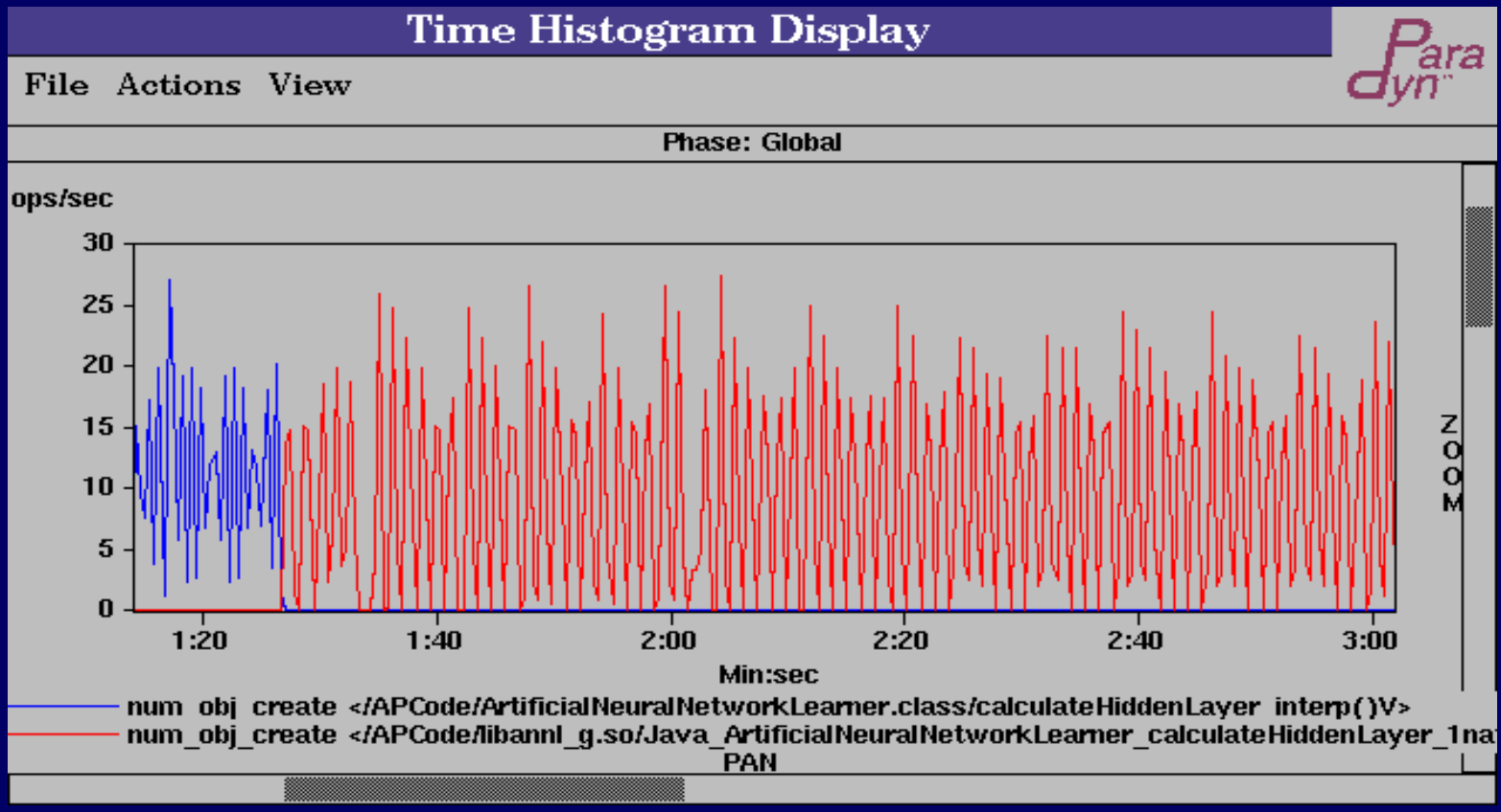


A method that doesn't benefit from run-time compilation



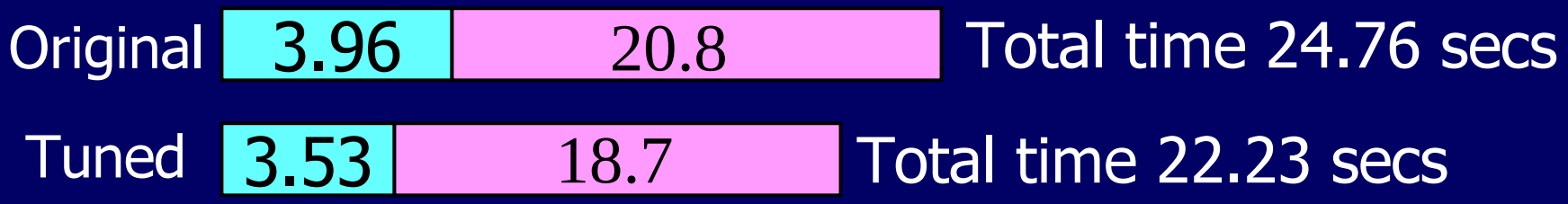
Why not?

VM still handles all memory management



How can we tune the Java AP?

- Remove some object creates
  - 10% improvement in method's execution time



- ExactVM's execution of the tuned AP
  - 10% improvement in total execution time
    (21.09 seconds vs. 18.97 seconds)

How can we tune the Java VM?

- Tune the VM routines responsible for handling object creates in the Java application
- Tune the dynamic compiler's run-time compiling heuristics
  - characteristics of method that make it a bad candidate?
  - incorporating profile data into the heuristic

Conclusions

- Paradyn-J provides data to easily determine how to tune application
  - measure AP byte-code and native code
  - measure VM interactions w/ AP native code
  - measure AP transformations
  - instrument unmodified binaries and .class files
- AP developers can see inside VM
- VM developers can characterize VM's performance in terms of AP code it runs