Profiling Dynamically Compiled Java

Tia Newhall

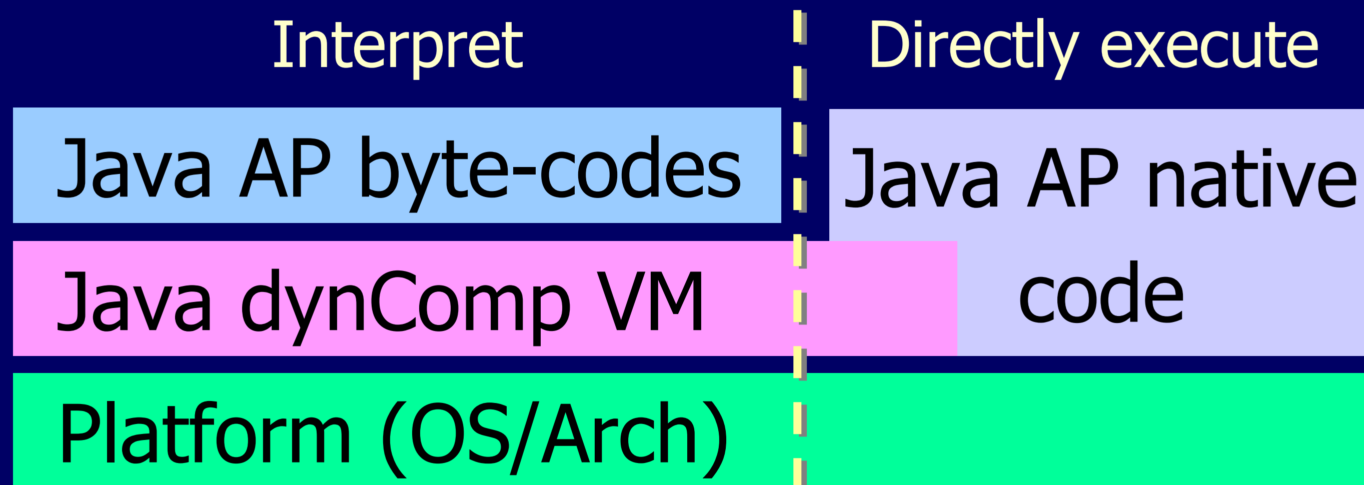
newhall@cs.wisc.edu

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

Why Java?

- Java is is slow, but it is being used for all kinds of things
  - Web-based computing
  - Meta-computing (Globus, Javelin, Charlotte)
  - High performance numeric applications (NPAC, JNT, JAMA)
  - Parallel computing (jPVM, JPVM, Java-MPI, HPJava, Titanium)
- Java VM's are getting faster
  - HotSpot as fast as equivalent C++

Dynamically Compiled Java



- Java application changes form at run-time
- Even in native code form, an application method interacts with Java VM

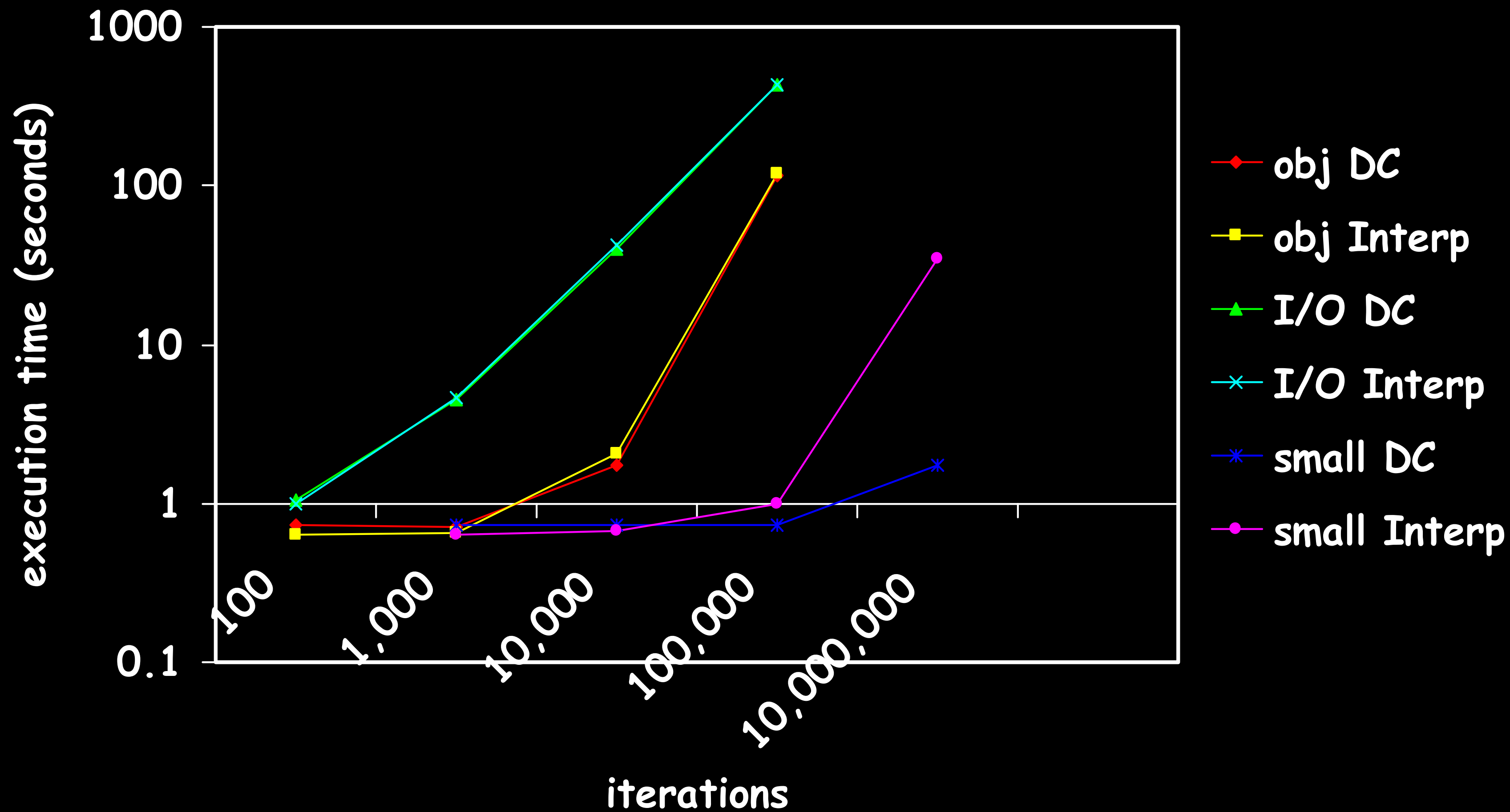
Performance measurement important…oh yeah, prove it

- Our hypothesis
  - dynamic compilation might not be the only answer
- Tested 3 application kernels on Sun's ExactVM dynamic compiler
  - Platform2 release of JDK
- ExactVM's run-time compiling heuristic:
  - if a method contains a loop, compile it immediately
  - else, wait until a method is called 15 times

Test Applications

- Application kernels test cases where we suspect dynamic compilation may not win
  - method's whose time not dominated by interpreting byte-code (I/O or synchronization)
  - method's whose native code form still has a lot of interaction with Java VM (object creates)
  - small method functions
- A mainloop method calls methods implementing one of the three cases

Results



What did we learn?

- There is something going on in this execution that we would like to see...
  - performance measures with native code form and byte-code form of a method function
    - did run-time compilation help? why not?
  - Java VM interactions with native code form of a method
    - what are these interactions?
    - how much do they affect the application's execution?

Paradyn-J

- Profiles dynamically compiled Java
  - simulate dynamic compilation
  - wrapper calls byte-code & JNI native versions
- Performance data that:
  - explicitly describes interactions between the VM and the Java application
  - associated with multiple execution forms of Java application methods
  - describes run-time costs of dynamically compiling a Java method

© 1999 Tia Newhall

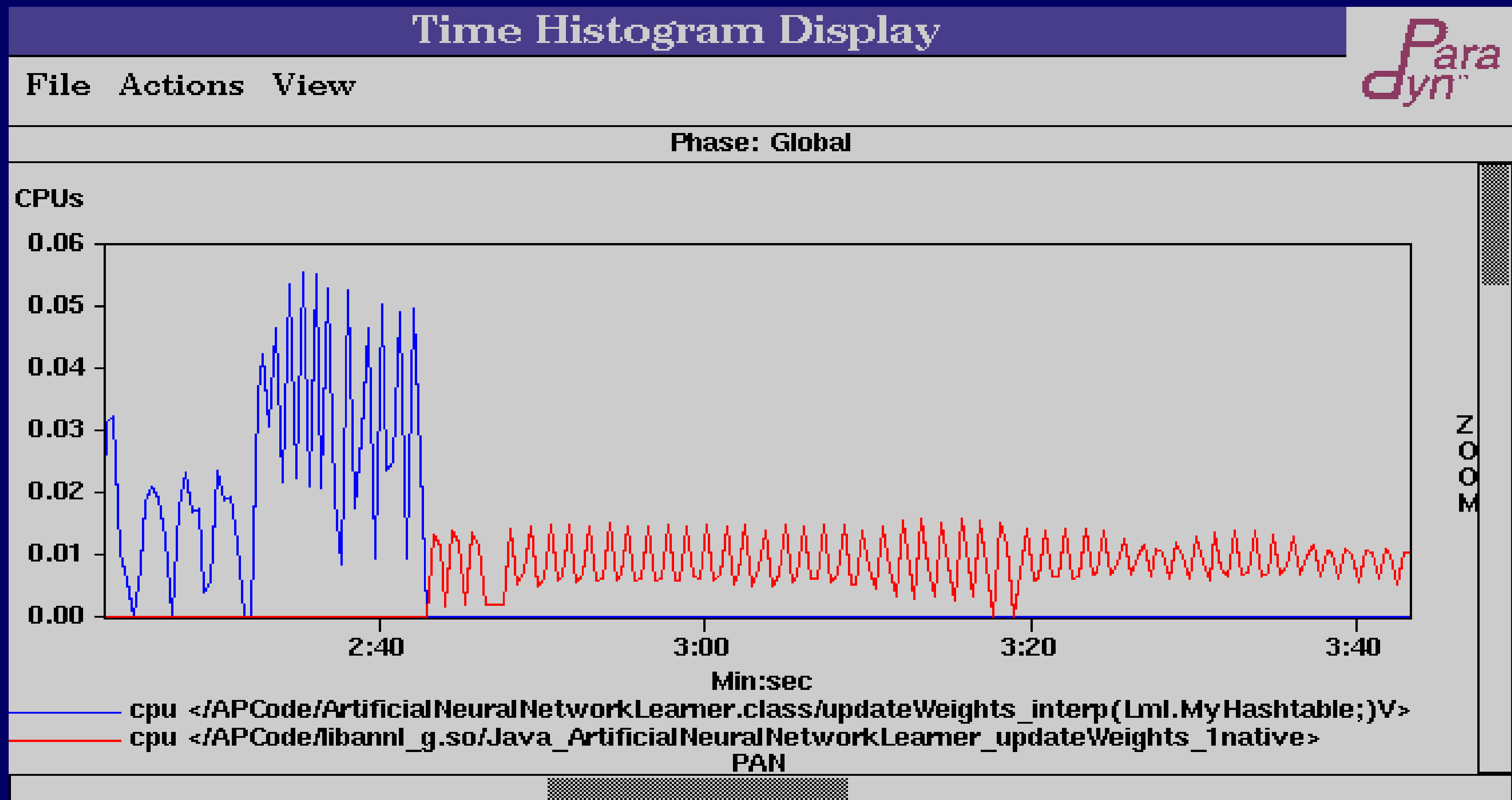
Let's see what we can do…

Method with object creates		I/O intensive method			Small methods		
	Byte-code		Native	Byte-code		Native	Byte-code
Total CPU	2.35	Total I/O time	5.65	0.37	CPU	4.9 µs	6.7µs
Obj. create overhead	1.57	Total CPU	0.01	0.04	Method call cost		2.5µs

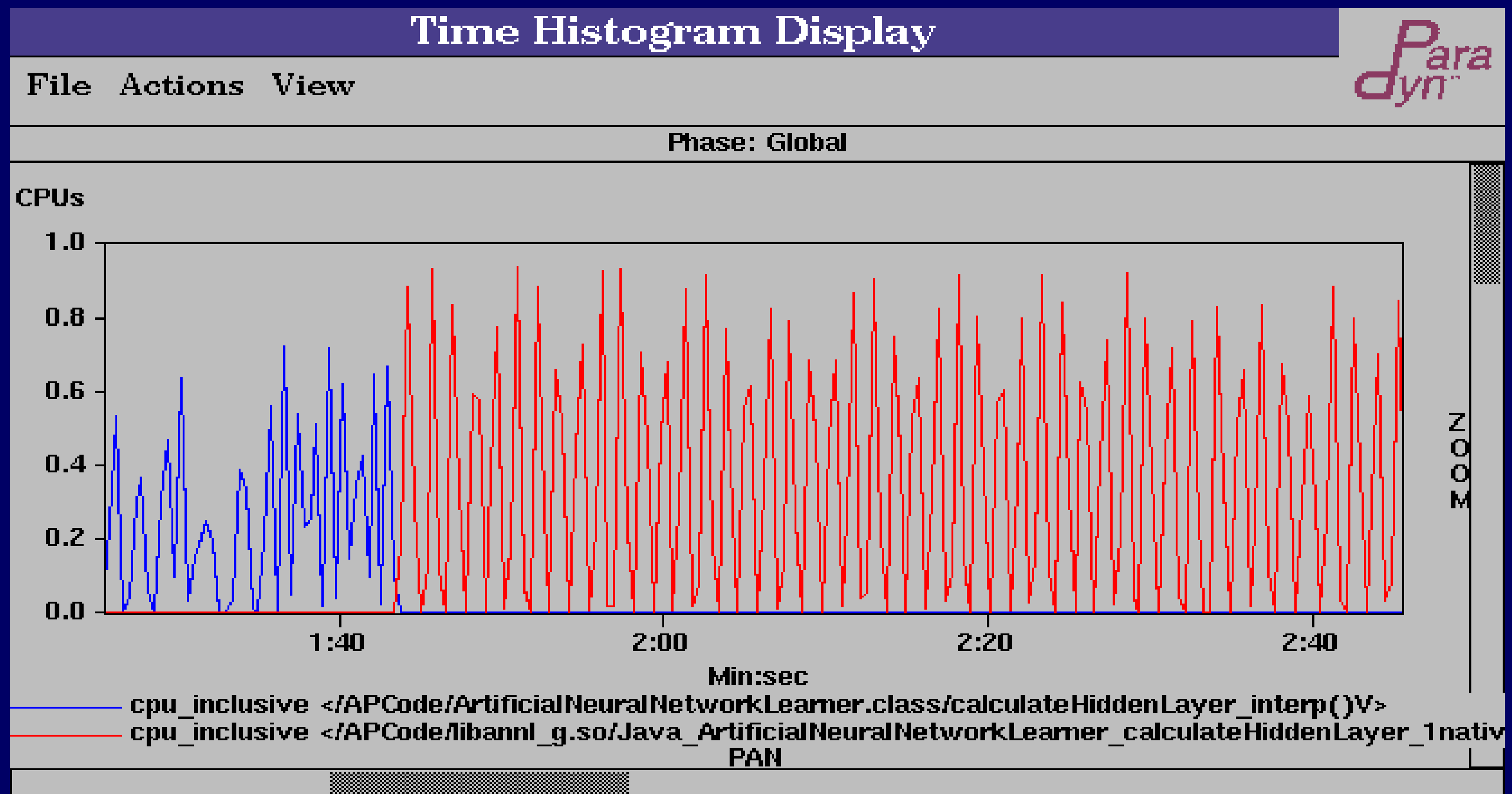
Performance tuning study

- Java neural network application

  23 classes and 15,800 lines of Java source

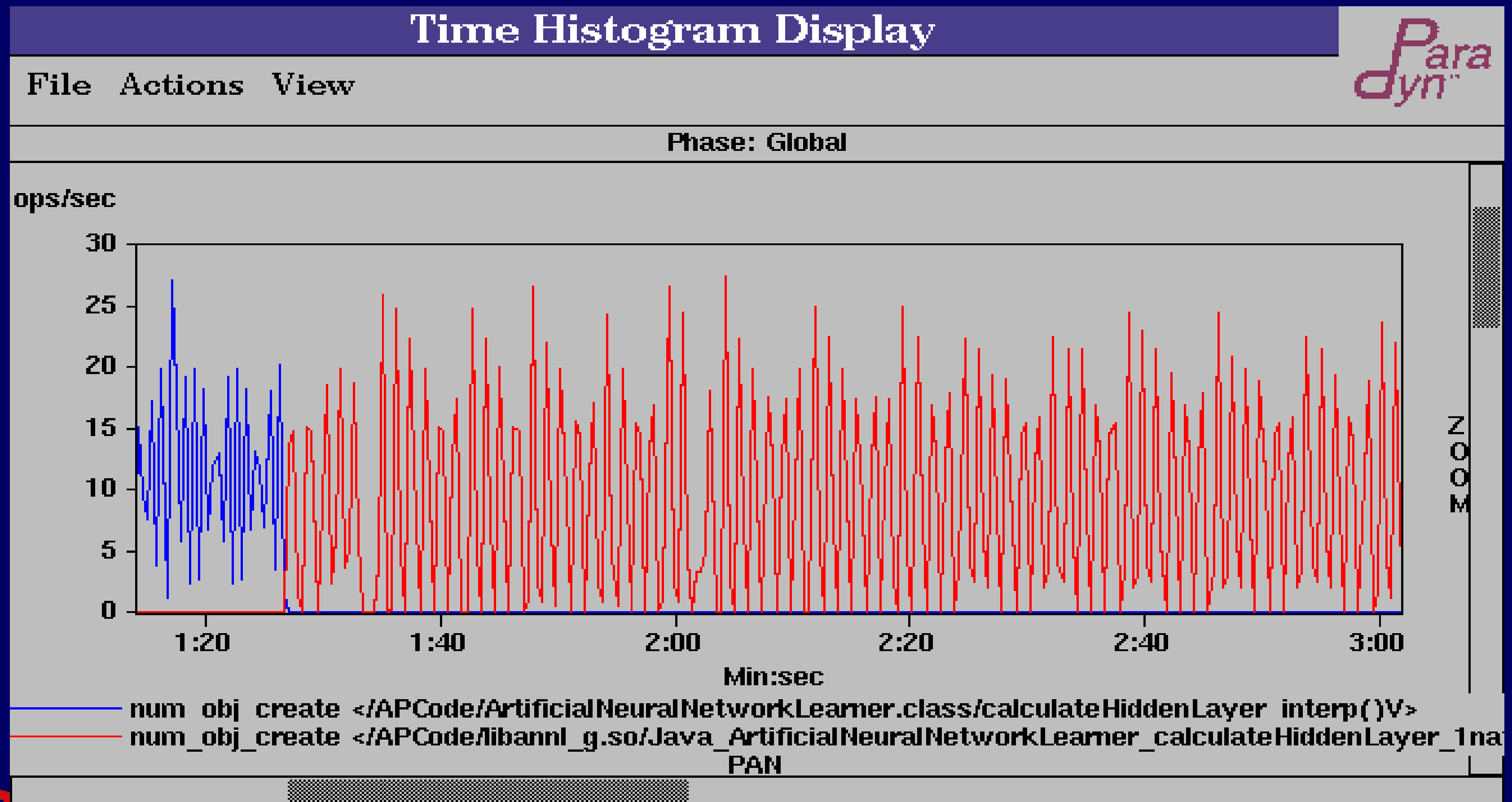


A method that doesn't benefit from run-time compilation



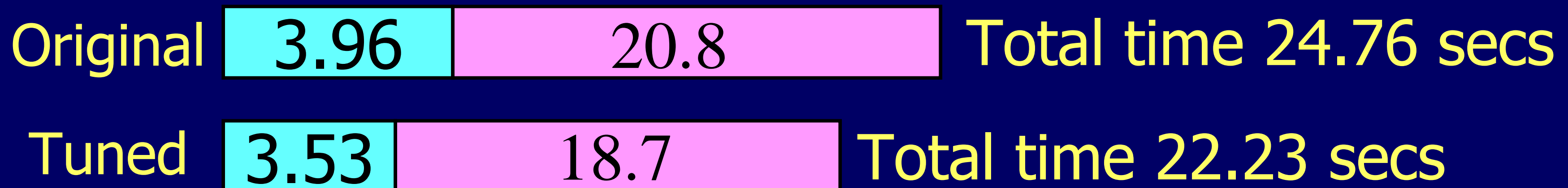
Why not?

- VM still handles all memory management



How can we use this data to tune the Java application?

- remove some object creates



⇨ improved method's performance by 10%

Applying tuning to a real dynamically compiled execution

- Run tuned version on real Java dynamic compiler (Sun's ExactVM)

original	tuned	change
21.09	18.97	10%

- Why does this make sense?
  - simulation adds extra overheads not in ExactVM
  - object creation overheads about the same

What about the 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 heuristics

Conclusions

- Java is here to stay
- More sophisticated VM's will ensure this
- Performance measurement of dynamically compiled Java is complicated
- Paradyn-J provides data that
  - lets us see inside the dynamic compiler to see how it executes the application
  - characterizes the VM's performance in terms of the application code it dynamically compiles

Paradyn™
© 1999 Tia Newhall