Next Generation VMs

Albuquerque

January 11, 2006

Schedule

9:00-10:00 Introduction, aims, overview
10:00-10:30 Coffee
10:30-12:00 Talks (Click, Grove, Lagergren, Vitek)
12:00-1:30 Lunch
1:30-3:00 Technical discussions
3:00-3:30 Coffee
3:30-5:00 Management model, future directions
5:00-6:00 Drinks
6:00 Dinner

Introductions

Note Takers?

Ground Rules

- This meeting is completely open
- Notes will be on the public record
- No confidential material to be divulged
- No one signs an NDA

Goal for Our Meeting

Establish a community effort to build an Apache-licensed *next generation* JVM

- Help establish/confirm directions
- Characterize ‘next generation’
- Share experiences
- Critique the approach
- Produce a public record of our ideas

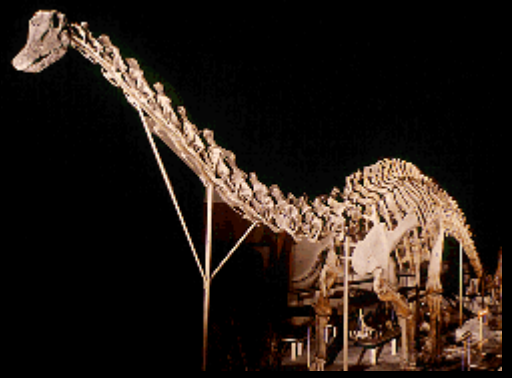
TODO

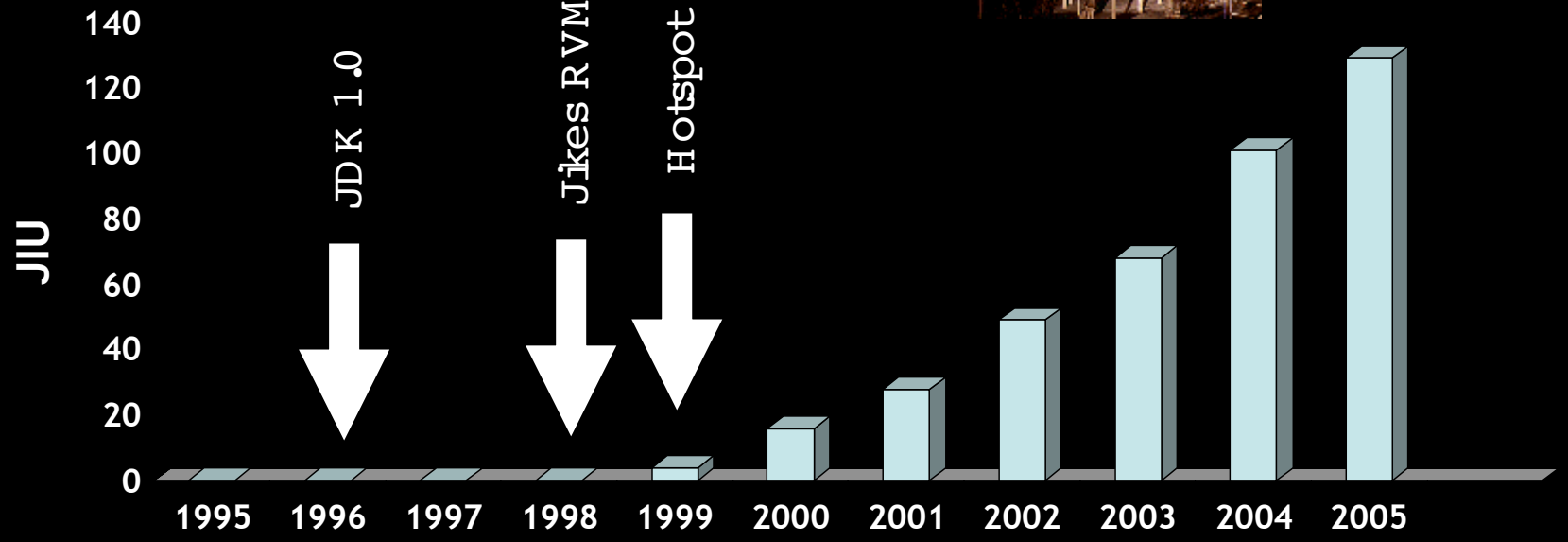
- We need a name!
  - Suggestions to me at end of this session
- Technical discussion points
  - Suggestions to me ASAP
- Management discussion points
  - Suggestions to me ASAP

Background

Background

- Apache Harmony
- Jikes RVM





NGVM Aims & Goals

The NGVM project will create an open source platform for developing product-quality JVMs and an environment for JVM innovation.

Our goal is to build the most innovation-friendly JVM development platform to date and from it, the most innovative product-quality JVMs. A product-quality JVM has outstanding performance, scalability and reliability.

The development philosophy of NGVM is to leverage its openness to draw on the most experienced members of the VM research and development community, from both industry and academia.

NGVM Priorities

The highest NGVM priorities are to be product-quality, innovative, and open.

Of these, the focus on innovation most distinguishes NGVM from existing JVMs (both proprietary and open). The priority of being *product-quality* means NGVM's goals include performance and robustness matching or exceeding that of commercial JVMs.

The priority of being *innovative* means NGVM will be designed to maximize flexibility and compose-ability. The priority of being *open* characterizes the NGVM development strategy and means that portability is one of NGVM's foremost goals.

NGVM Motivation

The highest performing JVMs of today rest on substantial code bases with long histories. The scenery has changed dramatically since the foundations for those VMs were laid. Architecture has changed dramatically (multicore, locality, stronger PMU support, etc). Implementation technology has advanced tremendously (optimizing JITs, feedback directed optimization, new GC algorithms, etc). The application landscape has changed considerably.

For all these reasons it is essential that the NGVM development approach start with a clean sheet and draw on the huge body of experience that has accumulated over this time. As a simple example, the core of Jikes RVM was built in 1998, only three years after Java became widely known. The changes in the Java landscape over the seven years since have been enormous; 130 research publications from the Jikes RVM project alone attest to this.

NGVM Need

NGVM targets all potential developers including those from industry and academia. The choice of the Apache license is intended to make NGVM friendly to both open and proprietary development on top of the NGVM platform.

The project also targets the research community, who require an open, high performance, innovation-friendly JVM. The emphasis on componentization encourages customization and experimentation, both open and proprietary.

The NGVM project will develop VMs for a wide range of *end users*, targeting the spectrum from embedded devices through the desktop to server applications.

NGVM Scope

NGVM will focus on developing a modular framework and foundation for VM implementation. Specifically, NGVM will develop a componentized VM core into which major components such as memory managers, JITs, and class-libraries can be plugged, allowing developers to use their own GCs, JITs and class libraries.

NGVM will therefore not begin with a focus on developing the JIT, GC, or class libraries. Furthermore, NGVM will focus on componentization within the VM core to allow sub-components such as schedulers, object models, etc, to be interchanged and independently developed. The primary design and implementation focus will therefore be on a) attacking the tension between performance and design flexibility, and b) in developing a strongly componentized VM core with a great deal of flexibility.

NGVM Approach

NGVM will create a JVM development framework from scratch. Its highest development priority will be to leverage a wealth of prior experience in VM development from the community, including strategic direction, design and implementation. Development will take place within the framework of the Apache community. The standard Apache development model places control squarely with the committers. The NGVM project will create a development model which draws guidance and expertise from individuals who are unable to be committers (either through tainting or other constraints). For example, NGVM may create an advisory board of non-committer expert advisors.

One take on all this…

Big Picture: Usage Models

- Industrial users
  - Basis for product and/or in-house R&D
- Researchers
  - High performance, flexible, credible
- Open source end users
  - A solid, high performance JVM

Big Picture: Code base

- VM Core
  - Product-quality & Apache-licensed
  - Componentized (intra & inter core)
  - Extensible & customizable by users
- Components (GC, JITs)
  - Product-quality & Apache-licensed
  - Extensible, customizable & *replaceable*

Examples: Product

1. Value-add: architecture & OS
   - Leverage high portability
   - 99% code base reuse
2. Value-add: compiler, dynamic optimization
   - Exploit componentization
   - 50% code base reuse

Examples: Research

1. Validate new architecture
   - Leverage highly portable, credible JVM
2. Evaluate cluster-based JVM
   - Exploit flexible code base
3. Explore new GC algorithms
   - Use flexible code base, debugging facilities

Big Picture: Management Model

- Developers (N people)
  - Commit code
- Advisory Board (~10 people)
  - Expert advisors
  - No formal powers
- Steering Committee (~3 people)
  - Guardians of project vision
  - Veto power over developers

Details: NGVM Prototype

- Java in Java (JiJ)
  - Software engineering advantages of Java
  - Eating one's own dog food…
- Portability
- 'Wedge' componentization (as in MMTk)
- Code persistence
  - Boot image, AOT, JIT persistence, code cache
- Isolates
  - MVM and JiJ app/user separation
- Small footprint
- Debugging capacity high priority
- Clean systems programming (in Java)

Names

Simple

Evocative

TODO

- Technical discussion points
  - Suggestions to me ASAP
- Management discussion points
  - Suggestions to me ASAP

Schedule

9:00-10:00 Introduction, aims, overview
10:00-10:30 Coffee
10:30-12:00 Talks (Click, Grove, Lagergren, Vitek)
12:00-1:30 Lunch
1:30-3:00 Technical discussions
3:00-3:30 Coffee
3:30-5:00 Management model, future directions
5:00-6:00 Drinks
6:00 Dinner

Talks

TODO

- Technical discussion points
  - Suggestions to me ASAP
- Management discussion points
  - Suggestions to me ASAP

Names

12 XVM (X=extreme, variable, extra, experimental…)
11 JIG
7 JVolution
9 Moxie/JMoxie
6 6 Habanero (excellent pepper compared to jalapeno)
6 6 Innovation VM (IVM)
3 4 Borneo
2 4 Serenity
1 4 Evolution VM
0 5 Melody
0 JEV

3 Nexus
3 Sand (From Chinese for 3rd gen, and implies ubiquity)
3 Sauron (One VM to rule them all!)
3 VM Bones (VM skeleton)
2 Javanate
2 Kopi Tiam (coffee house)
1 Delta (Change/innovate)
1 Eta
1 Jenerate
1 Trio
1 MarVel

Jeta
Jevolve
Lego VM
Kopi Luwak (outstanding coffee)
Scotch bonnet (excellent pepper compared to jalapeno)
Serendipity
Trident (3rd gen)
Agile VM
Bubbler
Carbonate
Cabernet
Flex VM
Gamma (3rd gen)
J16
Janus (Roman god of gates and doors, beginnings and endings)
Jet

Schedule

9:00-10:00 Introduction, aims, overview
10:00-10:30 Coffee
10:30-12:00 Talks (Click, Grove, Lagergren, Vitek)
12:00-1:30 Lunch
1:30-3:00 Technical discussions
3:00-3:30 Coffee
3:30-5:00 Management model, future directions
5:00-6:00 Drinks
6:00 Dinner

Technical Discussions

Systems Programming In Java

- **Intrinsic methods.** (‘`intrinsic`’ annotation on method)
  - Declaration is simply a prototype: the compiler provides the implementation.
- **Unboxed types.** (‘`unboxed`’ annotation on class)
  - Cannot be heap allocated (fields of objects (in object) or local variables (on stack))
  - Casting mechanism to cast “raw” memory (such as a stack frame or object header) as an unboxed type
  - Control the layout of the unboxed type (annotations)
  - Control which field of the unboxed type its reference points to (not necessarily the first) (annotations)
  - Operators may be defined for unboxed types (using annotations). Arguably:
    - operators must always be intrinsic (semantics are provided by the compiler), and/or
    - operators must be in a new name space “:+” rather than “+” (can’t be confused with existing operators)
- **Native types.** (new types?)
  - “*native int*” and “*native uint*” as primitive types, would allow clean(er) implementation of Address and Word types, necessary for systems programming.

(J)VM Kernel/User Separation

Minimizing TCB

Portability

- Arch
  - CISC/RISC
  - Register windows, or not
  - 64/32 bit words & endianness
  - Multi-processor (essential)
- Memory models
  - Cache coherence
  - Code patch: icache assumptions
- GC
  - Moving/non, read/write barriers, concurrent, parallel, pinning, interior pointers
- H/W (OS) Stacks
  - What direction do they grow?
  - Single or multiple piece
- OS
  - Threading model & priorities (are priorities available?)
  - Is address translation available?
  - Is memory contiguous?
  - CPU affinity
  - Page protection? (can we depend on it)

Object Models

- Need to support many object models
- Need to encapsulate and abstract over it
- Who determines it:
  - GC, locking, dispatch, hash, type ops
- Who uses it:
  - Interpreter/JIT, GC, locking, debugger, JVMTI et al, VM reflective (boot image, JNI), class libraries
- Need to support experimentation
  - Adding fields should be easy
- In long term we want to generate
  - ‘Gather’ specs and from this form a high level spec

Kernel/Library dependence

- How much of Java can we depend on?
  - In boot-strap, and more generally
  - Which version of JDK/JRE do we depend on?
- What constrains this?
  - Bootstrap issues, space constraints
- What do we implement ourselves?
  - Memory allocation, hash tables, basic IO, strings? …
- What are we going to avoid?
  - Floating point …
- Implementation strategy: make home-baked versions as compatible as possible

Support for Numerical Code

- If someone cares enough…..

Multiple Languages?

- Bytecodes are a problem
  - More expressive than Java, really difficult to work with…

Abstracting Over Perf Ctrs

Schedule

9:00-10:00 Introduction, aims, overview
10:00-10:30 Coffee
10:30-12:00 Talks (Click, Grove, Lagergren, Vitek)
12:00-1:30 Lunch
1:30-3:00 Technical discussions
3:00-3:30 Coffee
3:30-5:00 Management model, future directions
5:00-6:00 Drinks
6:00 Dinner

Management, Development & Governance

Thank you!

Schedule

9:00-10:00 Introduction, aims, overview
10:00-10:30 Coffee
10:30-12:00 Talks (Click, Grove, Lagergren, Vitek)
12:00-1:30 Lunch
1:30-3:00 Technical discussions
3:00-3:30 Coffee
3:30-5:00 Management model, future directions
5:00-6:00 Drinks
6:00 Dinner