Parley: Federated Virtual Machines

Perry Cheng, Dave Grove, Martin Hirzel, Rob O’Callahan and Nikhil Swamy
What is Parley?

- **Motivation**
  - Virtual machines (VMs) are increasingly important
  - Heterogeneity in languages, programming models, VMs seems inevitable
  - Strong desire to interoperate: cross-language and cross-VM

- **Scope of the Parley research project**
  - VM (and IDE) support for cross-language interoperation
  - How to structure VMs to increase flexibility, reusability, maintainability, etc.
Outline

- Three approaches for interoperability
- Parley in more detail
  - Basic scenario
  - Extensions
  - Current status
- Discussion
Interoperability via a Single VM

- Compile multiple source languages into a single VM target language
- VM really understands this common target language, not the various source languages

Advantages
- Leverage investment in VM infrastructure
- No cross-language interoperability issues (at the VM level)

Disadvantages
- Not every language can be translated without loss of fidelity
- Monolithic VM makes it harder to innovate/evolve (language & VM)
- Large deployed base of “legacy” VMs and programs that may rely on language semantics that get lost in translation
Loosely-Coupled Interoperability

- Program modified to talk to “foreign” languages via OS-level IPC
- Advantages
  - Allows VM heterogeneity; pick VM that best matches language (or legacy)
- Disadvantages
  - Programming model (can be partially alleviated via sophisticated tooling)
  - Performance (when VM crossings are frequent)
Parley: Federated Virtual Machines

- VMs are modified to interface to Parley interop library
- Advantages
  - Allows VM heterogeneity; pick VM that best matches language (or legacy)
  - Allows high-performance: single process ➔ cross-VM call can be lightweight
  - Programming model unchanged (assumes IDE also understands Parley interop)
- Disadvantages
  - VM modifications required (modest, but non-trivial)
Why do we think Parley is attractive?

- **VM heterogeneity**
  Languages with specialized requirements can interoperate without giving up (internally) on their own unique features
  Customized compiler, runtime, or type system
  Languages and VMs can evolve independently instead of in lock-step
  Multiple vendors can contribute

- **Footprint**
  Individual VMs can be smaller and simpler
  Supporting Parley should add minimal overhead to a VM

- **Interesting alternative to current technology (research problem)**
A Simple Parley Scenario

Executing a program written in language A that utilizes a component written in a “foreign” language Z

Diagram:
- Application (Language A)
- Component (Language Z)
- Interoperation
- Access “foreign” resource
- OS + CPU
- Parley
- VM
Summary of Parley Interop Layer

- **Coordination**
  - Register, create, manage, and destroy VM instances
- **Metadata (resources)**
  - Fairly generic notion of resource
  - Rely on attribute language to describe constructs within resources
    - Types: Reference, Values, Blob
    - Functions, slots
    - Presentational hints (constructor, accessor, etc)
- **Data**
  - Object model
  - Memory management
  - Auto-mapping of fundamental types (strings, primitives)
- **Control**
  - Object model
  - Calling conventions
  - Exceptions and stack walking
Extensions

- Optional richer Parley API that enables deep cross-VM integration
  - Direct invocation of foreign functions
  - Direct manipulation of foreign objects (pass by reference, not via proxy)
  - Cross-VM inheritance and interface implementation

- Key ingredients
  - Parley object model (specifies *some* of object model, not everything)
  - Cross VM-cooperation for GC
Parley Object Model

Reserve portion of “Class” at fixed offset from start

“Class” Ptr

Dispatch Table for A

Dispatch Table for B

Dispatch tables either point directly to code or contain “thunk” that does impedance matching
Optional Sharing of Common Components

- Some key subsystems could be common across multiple VMs
  - JIT optimizer and backends
  - Memory Management (GC)

- Common components
  - Reduce development cost
  - Reduce VM footprint
  - Facilitates optimizations
  - Cross-VM function calls with minimal thunks; cross-VM inlining
  - Cross-VM references ➔ coordination of GC (easier if same GC)
Parley Current Status

- Early stages of a research project (not product development)
- Defined and implemented prototype Parley interop layer
- Modifying a JVM and CLR (mono) to interface to Parley
- Looking for suggestions on an interesting VM (significantly different language model) as third target to stress interfaces and shake out assumptions
Discussion

- How viable is this architecture?
- What VMs would be interesting to include?
- Usage scenarios to evaluate strength and weaknesses of each potential approach?