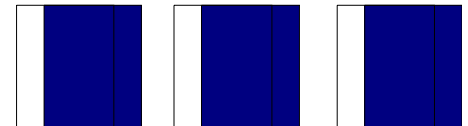




Jaguar: Enabling Efficient Communication and I/O in Java

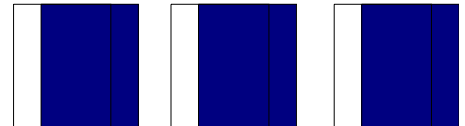
Matt Welsh and David Culler
UC Berkeley

Presented by David Hovemeyer



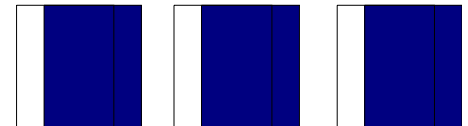
Outline

- ❑ Motivation
- ❑ How it works
 - ❑ Code mappings
 - ❑ External objects
 - ❑ Pre-serialized objects (PSO)
- ❑ Example: VIA
- ❑ Performance
- ❑ Conclusions



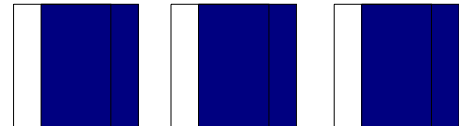
Motivation

- ❑ Use Java in high–performance computing
 - ❑ Big servers
 - ❑ Clusters
- ❑ Improving computation has been studied
 - ❑ I.e., JIT compilers, better garbage collectors, etc.
- ❑ But we also need high performance I/O
 - ❑ Including direct access to hardware



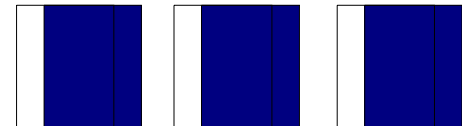
Motivation

- ❑ Jaguar could be used for a lot of things
 - ❑ Not just I/O
 - ❑ Some ideas at conclusion of talk



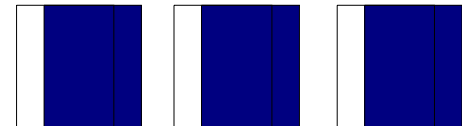
Why is Jaguar necessary?

- ❑ Can't we use native methods (JNI)?
- ❑ Native methods are really slow
 - ❑ JNI method calls 20x slower than equivalent C call
 - ❑ Memory copies between Java and C code 4 – 160 times slower than memcpy()
- ❑ Why is JNI so slow?
 - ❑ Too much copying? Interaction with GC?
 - ❑ Could it be fixed?



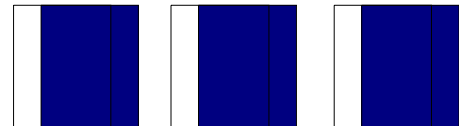
Why is Jaguar necessary?

- ❑ Native methods not just a performance problem
 - ❑ Too low level, error prone
 - ❑ Java is (arguably) easier to write and maintain



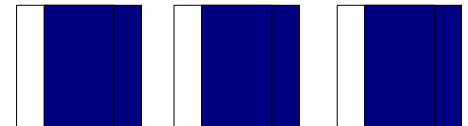
Jaguar

- ❑ Jaguar: Java Access to Generic Underlying Architectural Resources
- ❑ Present underlying OS and hardware resources as Java objects!
- ❑ Take advantage of some of the safety guarantees offered by Java: bounds checking, can't access arbitrary memory



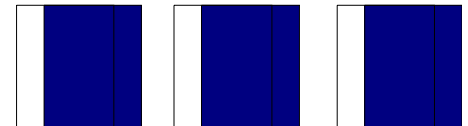
How it works

- ❑ Main ideas:
 - ❑ **Code mappings:** translate small sequences of bytecode into machine code
 - ❑ **External objects:** allow (controlled) access to memory outside Java heap
 - ❑ **Pre-serialized objects:** efficiently maintain external representation of objects in memory for fast I/O



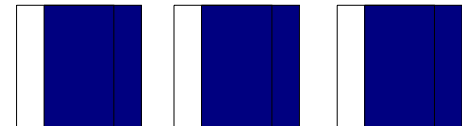
Code mappings

- ❑ Translate a sequence of bytecode instructions into machine language
 - ❑ Example: translate a method call into a loop to poll a memory-mapped I/O register
- ❑ Translation done by the JIT compiler
- ❑ Mappings must be done statically
 - ❑ So runtime types are not taken into account (i.e., for `invokevirtual`)



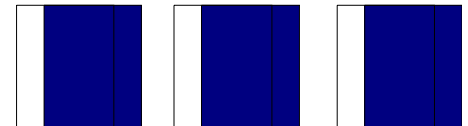
External objects

- ❑ External objects: access to memory outside the Java heap
- ❑ Examples:
 - ❑ Memory-mapped I/O buffers
 - ❑ Shared memory
 - ❑ I/O devices
- ❑ Accessed as “ordinary” Java objects
 - ❑ Presumably with the help of code mappings?



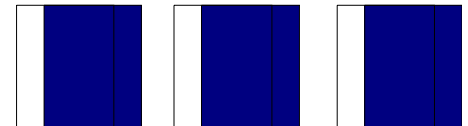
Pre-serialized objects

- ❑ Serialization is slow
 - ❑ Too slow for fast I/O channels
- ❑ Maintain objects in external form in memory
 - ❑ So they can be blasted from/to I/O channels quickly
 - ❑ Could be used for fast argument marshalling and unmarshalling in RMI (?)
- ❑ More limited than normal serialization
 - ❑ Object references may not go outside a particular “container”



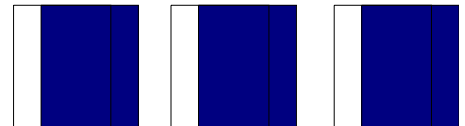
Example: VIA

- ❑ VIA: Virtual Interface Architecture
 - ❑ A standard for user-level networking
 - ❑ Applications directly access and manipulate tx and rx queues
 - ❑ Low-level: no flow control, etc.
- ❑ Intended application seems to be clusters (?)



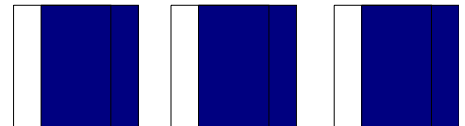
Berkeley VIA

- ❑ Berkeley VIA uses 1.2 Gb/s Myrinet
- ❑ Hardware interface:
 - ❑ tx and rx descriptor queues
 - ❑ tx and rx “doorbells”
 - ❑ tx and rx buffers in pinned memory
 - ❑ dedicated onboard processor



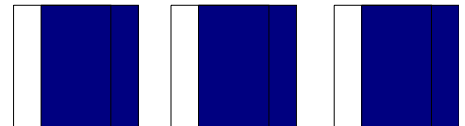
Berkeley VIA

- ❑ To transmit:
 - ❑ Construct a descriptor of the tx buffer(s)
 - ❑ Write to the tx doorbell registers
- ❑ To receive:
 - ❑ Constructor a descriptor of the rx buffer(s)
 - ❑ Poll the rx doorbell register



JaguarVIA

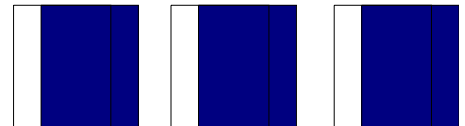
- ❑ JaguarVIA: Java interface to VIA
- ❑ Presented as Java classes:
 - ❑ VIA_VI: represents a VIA interface
 - ❑ VIA_Descr: represents a VIA descriptor
 - ❑ VIA_Databuffer: registered memory buffer



JaguarVIA

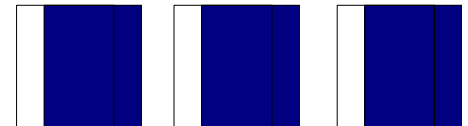
- Example method (from the paper):

```
public class VIA_VI {  
    public int VipPostSend(VIA_Descr descr){  
        while( TxDoorbell.isBusy() ) /*spin*/;  
        TxDoorbell.set(descr);  
        return VIP_SUCCESS;  
    }  
}
```



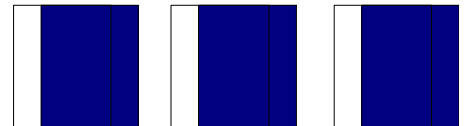
JaguarVIA

- ❑ Implementation of code mappings
- ❑ **`TxDoorbell.isBusy()`**
 - ❑ Reads dword from memory handle register
 - ❑ Sets boolean value in `%edx`: was memory handle register == 0
- ❑ **`TxDoorbell.set(VIA_Descr descr)`**
 - ❑ Writes handle of data buffer to mem handle register
 - ❑ Writes address of descriptor to descriptor register



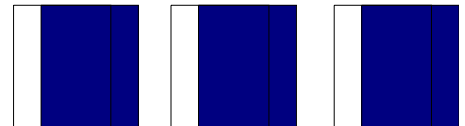
JaguarVIA performance

- ❑ Almost identical to C code for round trip time
- ❑ Within a few % of C code for raw bandwidth
 - ❑ Within 1% for large messages



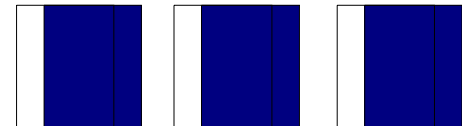
PSO performance

- ❑ Small performance hit to read/write PSO fields
- ❑ 100x faster than normal Java serialization for round trip time through VIA
- ❑ Around 33% slower than raw VIA



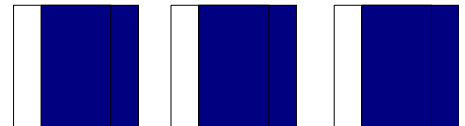
Conclusions

- ❑ My opinions only!
- ❑ Good things about Jaguar
 - ❑ Simple, fast, flexible
 - ❑ Seems like a good fit with the language
- ❑ Criticisms of Jaguar?
 - ❑ How safe are code mappings? How difficult to write?
 - ❑ PSO mechanism seems limited – no refs outside of container. How useful in practice?



Conclusions

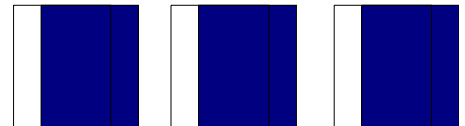
- ❑ Jaguar shows that resources (including hardware) can be allocated and protected within the Java language and runtime
 - ❑ Without relying on the operating system!
 - ❑ Maybe we don't really need an OS!



New I/O APIs for Java

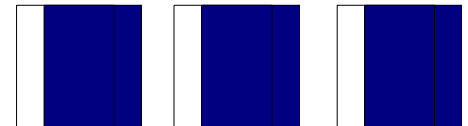
Java Specification Request #000051

Mark Reinhold, Dan Kegel, Doug Lea, Matt Welsh,
et. al.



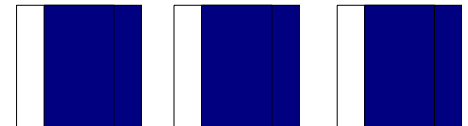
What's wrong with old I/O APIs?

- ❑ Not scalable
 - ❑ No asyc I/O or polling. Only directly supports thread-per-connection.
- ❑ Various performance problems
 - ❑ BufferedInputStream is synchronized!
 - ❑ Too much layering / filtering / copying
 - ❑ What else?
- ❑ Lacks some features
 - ❑ No regexps, limited output formatting



New I/O APIs

- ❑ Scalable – better support for servers, SMP
- ❑ Minimally synchronized
 - ❑ Caller responsible for synchronizing when needed
- ❑ Fast buffered binary I/O
 - ❑ Including memory–mapped files
 - ❑ Use Jaguar to implement?
- ❑ Fast character I/O with regexps and printf()–style formatting



New I/O APIs

- ❑ Better support for character set conversion
- ❑ Richer exception hierarchy
- ❑ Better filesystem support
- ❑ Enhanced kitchen sink functionality

