Packed Objects

Smarter software for a smarter planet
Important Disclaimers

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHilst EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.

ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.

IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.

IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS
Problem
Problem? What problem?

- JNI just isn’t a great way to marshal data
- Locality in Java can matter (e.g., JEP 142)
- Existing native and data placement stories aren’t very good
- In many cases, legacy systems exist – the interop is just terrible

- So we want something that integrates well with the Java language and helps us…
Native Access

Fighting the Java/Native interface
Everything is an Object

Hash → Array → Entry → (object)

- Hash: table
- Array: object field / data
- Entry: key
- (object): value

Object header
Object field / data
Everything is an Object

- Hash
- Array
- Entry
- (object)
- (object)

- Object header
- Object field / data
Everything is an Object

Hash → Array → Entry → (object)

- Hash: Table
- Array: Object header
- Entry: Key, Value
- (object): Object header, Object field / data
Everything is an Object
Field ordering has performance implications

JVM can potentially reorder your fields for you
Establishing Goals

- On heap / off heap seamless referencing of data
- Ability to do away with headers
- Ability to bring related objects close together

- This actually sounds a lot like C structure types

```c
struct Address {
    char[4] addr;
    short port;
};

struct Header {
    struct Address src;
    struct Address dst;
};
```

- Packed Objects!
Basics
Packed Objects: Under the covers

```
int x
int y
```

Diagram:
- Object header
- Object field / data

```
aPoint
```

```
Object header
```

```
Object field / data
```

```
int x
int y
```
Packed Objects: Under the covers

```
int y
int x
```

```
Object header
Object field / data
```

```
aPoint
int x
int y
```

```
aPackedPoint
target
offset
int x
int y
```
Packed Objects: Under the covers

```
int x
int y
```

```
Object header
Object field / data
```

```
int y
int x
```

```
aPoint
aPackedPoint
```

```
int x
int y
```

```
target
offset
int x
int y
```
Packed Objects: In Practice

**Diagram:**
- **aLine**
  - Point s
  - Point e
- **aPoint**
  - int x
  - int y
  - int x
  - int y

Legend:
- Object header
- Object field / data
Packed Objects: In Practice

- **aPoint**
  - int x
  - int y

- **aLine**
  - Point s
  - Point e

- **aPackedLine**
  - target
  - offset
  - int x
  - int y
  - int x
  - int y

Symbols:
- Object header
- Object field / data

© 2013 IBM Corporation
Packed Objects: In Practice

- `aLine`: `Point s` and `Point e`
- `aPoint`: `int x` and `int y`
- `aPackedLine`:
  - `target`
  - `offset`
    - `int x`
    - `int y`
    - `int x`
    - `int y`
- `aPackedPoint s` and `aPackedPoint e`
@Packed
final class PackedPoint extends PackedObject {
    int x;
    int y;
}

@Packed
final class PackedLine extends PackedObject {
    PackedPoint s;
    PackedPoint e;
}
Packed Objects: In Practice

```
int y
int x
aPoint
Point s
Point e
aLine

int x
int y
aPoint

int x
int y
aPoint

int y
int x
aPoint

int y
int x
aPoint

int y
int x
int y
aPackedLine

target
offset
int x
int y
int x
int y

Object header
Object field / data
```
Packed Objects: In Practice

- **aPoint**
  - int x
  - int y

- **aLine**
  - Point s
  - Point e

- **aPackedLine**
  - target
  - offset
  - int x
  - int y
  - int x
  - int y

- **aPackedLine.e**
Packed Objects: In Practice

```
int y
int x

Object header
Object field / data

aLine
Point s
Point e

aPoint

aPoint

aPackedLine
offset
target
int x
int y
int x
int y

aPackedPoint

aPackedLine.e
```
Packed Objects: In Practice with Arrays

anArrayOfPackedPoints

aPackedPoint

Object header
Object field / data

target
offset
int x
int y
int x
int y
int x
int y

...
Packed Objects: In Practice with Native Access

Java

Native

```
struct Point {
    int x;
    int y;
}
```
Packed Objects: In Practice with Native Access

Java

```java
@Packed
final class PackedPoint extends PackedObject {
    int x;
    int y;
}
```

Native

```c
struct Point {
    int x;
    int y;
}
```
Packed Objects: In Practice with Native Access

@Packed
final class PackedPoint extends PackedObject {
    int x;
    int y;
}

struct Point {
    int x;
    int y;
}
Advantages
Lets Build Something in C!

- Nested substructures
- Compact
- Alignment

```c
struct Address {
    char[4] addr;
    short port;
}

struct Header {
    struct Address src;
    struct Address dst;
}
```
Let’s Build the Same “Something” in Java!

- Headers
- Locality
- Alignment
What does the Java code look like under the covers?

```java
if (header.dst.addr[0] == (byte)192) {
    // ...
}
```

**Bytecodes:**
- `aload1`
- `getfield Header.dest LAddress;`
- `getfield Address.addr [B`
- `iconst0`
- `baload`
- `bipush 192`
- `ificmpeq ...`

**JIT (32 bit):**
- `mov EBX, dword ptr -4[ECX]`  // load temp1
- `mov EBX, dword ptr 8[EBX]`  // load dest
- `mov EBX, dword ptr 4[EBX]`  // load addr
- `movsx EDI, byte ptr 8[EBX]`  // array[0]
- `cmp EDI, 192`

- From a code point of view, this isn't terrible...
What if we did this with Packed Objects?

- The Java code is pretty clean… and a pretty good result!

```java
@Packed
final class Address extends PackedObject {
    PackedByte[[4]] addr;
    short port;
}

@Packed
final class PacketHeader extends PackedObject {
    Address src;
    Address dest;
}
```
Ok, what about the code under the covers?

```java
if (header.dst.addr[[0]] == (byte)192) {
    // ...
}
```

**Bytecodes:**
- `aload1`
- `getfield PackedHeader.dest LAddress;`
- `getfield Address.addr [B iconst0 baload bipush 192 ificmp eq ...`

**JIT (32 bit):**
- `mov EBX, dword ptr -4[ECX] // load temp1`
- `mov EAX, dword ptr 4[EBX] // load target`
- `mov EDX, dword ptr 8[EBX] // load offset`
- `lea EBX, dword ptr [EAX + EDX]`
- `movsx EDI, byte ptr 0[EBX] // array[0]`
- `cmp EDI, 192`

- Bytecodes don’t change… JIT code is pretty good too!
What about native access?

How do we implement this normally?
JNI implementation

```java
public class PackedHeader {
    private long pointer;

    public byte[] getSourceAddress() { return getSourceAddressImpl(pointer); }
    public short getSourcePort() { return getSourcePortImpl(pointer); }
}

JNIACALL jshort Java_pkg_PackedHeader_getSourcePort(JNIEnv* env, jobject recv, jlong pointer) {
    struct PacketHeader* header = (struct PacketHeader*)pointer;
    return (jshort)header->src.port;
}

JNIACALL jbyteArray Java_pkg_PackedHeader_getSourceAddress(JNIEnv* env, jobject recv, jlong pointer) {
    struct PacketHeader* header = (struct PacketHeader*)pointer;
    jbyteArray result = (*env)->NewByteArray(env, 4);
    (*env)->SetByteArrayRegion(env, result, 0, 4, &(header->src.addr));
    return result;
}
```

- Usual “stash pointers in long types” tricks
- JNI costs tend to be high
JNI implementation

```java
public class PackedHeader {
    private long pointer;

    public byte[] getSourceAddress() { return getSourceAddressImpl(pointer); }
    public short getSourcePort() { return getSourcePortImpl(pointer); }
}

JNI CALL jshort Java_pkg_PackedHeader_getSourcePort(JNIEnv* env, jobject recv, jlong pointer) {
    struct PacketHeader* header = (struct PacketHeader*)pointer;
    return (jshort)header->src.port;
}

JNI CALL jbyteArray Java_pkg_PackedHeader_getSourceAddress(JNIEnv* env, jobject recv, jlong pointer) {
    struct PacketHeader* header = (struct PacketHeader*)pointer;
    jbyteArray result = (*env)->NewByteArray(env, 4);
    (*env)->SetByteArrayRegion(env, result, 0, 4, &(header->src.addr));
    return result;
}
```

- Usual “stash pointers in long types” tricks
- JNI costs tend to be high
Unsafe implementation

```java
class PackedHeader {
    private Unsafe unsafe;
    private long pointer;
    private static final int SRC_ADDR_OFFSET = 0;
    private static final int SRC_PORT_OFFSET = 4;
    private static final int DEST_ADDR_OFFSET = 8;
    private static final int DEST_PORT_OFFSET = 12;

    public short getSourcePort() { return unsafe.getShort(pointer + SRC_PORT_OFFSET); }
    public byte[] getSourceAddress() {
        byte[] result = new byte[4];
        unsafe.copyMemory(null, pointer + SRC_ADDR_OFFSET, result, 0, 4);
        return result;
    }
}
```

- You shouldn’t be here
- Keeping your indices straight is never fun
DirectByteBuffer implementation

```java
class PackedHeader {
    private ByteBuffer buffer;
    private static final int SRC_ADDR_OFFSET = 0;
    private static final int SRC_PORT_OFFSET = 4;
    private static final int DEST_ADDR_OFFSET = 8;
    private static final int DEST_PORT_OFFSET = 12;

    public short getSourcePort() { return buffer.getShort(SRC_PORT_OFFSET); }
    public byte[] getSourceAddress() {
        byte[] result = new byte[4];
        buffer.get(result, SRC_ADDR_OFFSET, 4);
        return result;
    }
}
```

- No extra JNI to write (this is good)
- Still playing the indices game
PackedObject answer

```java
final class PacketHeader extends PackedObject {
    Address src;
    Address dest;

    public short getSourcePort() { return src.port; }
    public PackedByte[] getSourceAddress() { return src.addr; }
}
```

- Looks like natural Java code
- Foregoes JNI
- Same type capable of on-heap representation
Challenges
What does `aPackedLine == anotherPackedLine` mean?

→ The data is what really matters
Synchronization

Java

```
offset
target

aPackedLine
```

Native

```
int x
int y
int x
int y

anotherPackedLine
```

offset
target

Ø

Ø
Synchronization

Thread T1:
synchronized (aPackedLine) {
aPackedLine.e.x = 10;
}

Thread T2:
synchronized (anotherPackedLine) {
anotherPackedLine.e.x = 13;
}
Finalization

```
int y
int x
```

```
offset
target
aPackedPoint
```

```
offset
target
int x
int y
int x
int y
aPackedLine
```
Finalization

```
int y
int x
int y
int x
```

```
offset
target
aPackedPoint

offset
target
int x
int y
int x
int y
```

```
aPackedLine
```
Finalization

```
int y
int x
```

```
offset
target
```

```
int x
int y
```

```
offset
target
```

Finalization?
Finalization

```
int y
int x
int y
int x
```

Finalization?

```
target
offset
int x
int y
int x
int y
```

Finalization?

```
target
offset
```
Finalization

Java

<table>
<thead>
<tr>
<th>aPackedLine</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
</tr>
<tr>
<td>offset</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>anotherPackedLine</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
</tr>
<tr>
<td>offset</td>
</tr>
</tbody>
</table>

Native

<table>
<thead>
<tr>
<th>int x</th>
</tr>
</thead>
<tbody>
<tr>
<td>int y</td>
</tr>
<tr>
<td>int x</td>
</tr>
<tr>
<td>int y</td>
</tr>
</tbody>
</table>

Finalization?
aPackedLine.s = aPackedPoint;
Nested Data Structures

- Base types do share the same assignment operator
- Helps convey `aPackedLine == anotherPackedLine` as meaningless

```
aPackedLine.s := aPackedPoint;
```
Field Initialization

```java
@Packed
final class PackedPoint extends PackedObject {
    int x;
    int y;

    PackedPoint(int x, int y) { … }
}

@Packed
final class PackedLine extends PackedObject {
    PackedPoint s;
    PackedPoint e;

    PackedLine(int sx, int sy, int ex, int ey) { … }
}
```

No no-argument constructor

Implicitly instantiates PackedPoint objects for s & e fields
Field Initialization

@Packed
final class PackedPoint extends PackedObject {
    int x;
    int y;

    void init(int x, int y) {
        this.x = x;
        this.y = y;
    }

    PackedPoint(int x, int y) {
    }
}

@Packed
final class PackedLine extends PackedObject {
    PackedPoint s;
    PackedPoint e;

    PackedLine(int sx, int sy, int ex, int ey) {
        s.init(sx, sy);
        e.init(ex, ey);
    }
}

Advanced
Modeling Native Data Pointers

```c
struct BinaryTreeNode {
    struct BinaryTreeNode* left;
    struct BinaryTreeNode* right;
    // data
};
```

Java

```
aBinaryTreeNode
```

Native

```
*left
*right
<data>
```

```
*left
*right
<data>
```

```
*left
*right
<data>
```
Modeling Native Data Pointers

```
struct BinaryTreeNode {
    struct BinaryTreeNode* left;
    struct BinaryTreeNode* right;
    // data
}
```

Java

```
apBinaryTreeNode.right
```

Native

```
apBinaryTreeNode
```

offset

target

Ø

*left

*right

<data>

*left

*right

<data>

*left

*right

<data>

...
Modeling Native Data Pointers

```c
struct BinaryTreeNode {
    struct BinaryTreeNode* left;
    struct BinaryTreeNode* right;
    // data
}
```

```
32b or 64b?
```
Modeling Native Data Pointers

@Packed
class BinaryTreeNode extends PackedObject {
    @NativePointer BinaryTreeNode left;
    @NativePointer BinaryTreeNode right;
    // data
}

- Annotation to mark a field as a native pointer (rather than a Java one)
- Enhance `getfield/putfield` to recognize
- Restrict for security reasons
- Unmanaged pointers (no GC involvement)
Modeling Native Data Pointers

```
aBinaryTreeNode
  target
  offset
  right
  data
```

```
aBinaryTreeNode.right
```

**Java**

**Native**
Modeling Native Data Pointers

Java

```
aBinaryTreeNode
```

Native

```
aBinaryTreeNode
```

```java
aBinaryTreeNode.
```
Alignment

@Packed
final class Address
extends PackedObject {
    PackedByte[[4]] addr;
    short port;
}

@Packed
final class PacketHeader
extends PackedObject {
    Address src;
    Address dest;
}

```
@Packed
final class Address
extends PackedObject {
    PackedByte[[4]] addr;
    short port;
}

@Packed
final class PacketHeader
extends PackedObject {
    Address src;
    Address dest;
}
```
Alignment

@Packed
final class Address
extends PackedObject {
    PackedByte[[4]] addr;
    short port;
}

@Packed
final class PacketHeader
extends PackedObject {
    Address src;
    Address dest;
}

- Which is the correct default behavior?
- How do you get the alternate if that’s what you want?
Alignment

class A {
    int i;
    short s;
    short padding; // align
    long l;
}

class A {
    int i;
    short s;
    @Align long l;
}
Alignment

- Padding isn’t quite right in the context of nested structures...

```java
@Packed
final class Address extends PackedObject {
    PackedByte[[4]] addr;
    short port;
}

@Packed
final class PacketHeader extends PackedObject {
    @Align Address src;
    @Align Address dest;
}
```

```java
class A {
    int i;
    short s;
    short padding; // align
    long l;
}

class A {
    int i;
    short s;
    @Align long l;
}
```
@Packed
final class SimpleValue extends PackedObject {
  int value;
}

Java

Native

- Provide a field annotation @BigEndian (and @LittleEndian)
Possibilities
Let’s look at transferring data

Heap

A

B

C
Let’s look at transferring data
Let’s look at transferring data
Let’s look at transferring data

Heap

Remote Transfer
PackedObjects could help…
PackedObjects could help…
PackedObjects could help…

Heap

Remote Transfer
Making the data transfer easier…

Heap

Remote Transfer
Making the data transfer easier…

Heap

Remote Transfer

Specialized Heap Area
Making the data transfer easier…

Heap

Remote Transfer

Specialized Heap Area
Making the data transfer easier…

Heap

Remote Transfer

Specialized Heap Area
Making the data transfer seamless

Heap

Remote Transfer

Specialized Heap Area
Making the data transfer seamless
Making the data transfer seamless

Heap

Packed

Remote Transfer

Specialized Heap Area
Questions?
References

- **Get Products and Technologies:**
  - IBM Java Runtimes and SDKs:
  - IBM Monitoring and Diagnostic Tools for Java:

- **Learn:**
  - IBM Java InfoCenter:
    - http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/index.jsp

- **Discuss:**
  - IBM Java Runtimes and SDKs Forum:
Copyright and Trademarks

© IBM Corporation 2013. All Rights Reserved.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., and registered in many jurisdictions worldwide.

Other product and service names might be trademarks of IBM or other companies.

A current list of IBM trademarks is available on the Web – see the IBM “Copyright and trademark information” page at URL: www.ibm.com/legal/copytrade.shtml