Optimizing R VM: Allocation Removal and Path Length Reduction via Interpreter-level Specialization

Haichuan Wang\textsuperscript{1}, Peng Wu\textsuperscript{2}, David Padua\textsuperscript{1}

\textsuperscript{1} University of Illinois at Urbana-Champaign \textsuperscript{2} Huawei U.S. Research
Outline

- R Background
- GNU R VM and Performance Analysis
- Our Solution – ORBIT (Optimized R Byte-code InterpreTer)
- Performance Evaluation
- Conclusion
R Background

- **R language**
  - Dynamic Scripting Language, used in statistics domain
  - Origin from S language of Bell Lab

- **R GNU Virtual Machine**
  - The reference R implementation, maintained by about 20 people

- **The language for data analytics in the age of Big Data**

Tool Used By Competitors in Data Analytics Competitions at Kaggle.com

![Bar chart showing popularity of R and other tools](http://r4stats.com/articles/popularity/)
Different R Programming Styles

Type I: Looping Over Data

```r
for (j in 1:500) {
  for (k in 1:500) {
    jk <- j - k;
    b[k,j] <- abs(jk) + 1
  }
}
```

(1) ATT bench: creation of Toeplitz matrix

Type II: Vector Programming

```r
males_over_40 <- function(age, gender) {
  age >= 40 & gender == 1
}
```

(2) Riposte bench: a and g are large vectors

Type III: Native Library Glue

```r
a <- rnorm(2000000);
b <- fft(a)
```

(3) ATT bench: FFT over 2 Million random values
Performance Issues with Type I (Loop) R Programs

- **Speed**

  ![Bar chart showing slowdown of R on the Shootout benchmarks relative to C and CPython](chart.png)

  - Slowdown of $R^1$ on the Shootout benchmarks relative to C and CPython

- **Memory Consumption/Allocation**

  ```r
  r <- 0;
  for( i in 1:1000000) { #1M
    r <- r + i;
  }
  print(r);
  ```

<table>
<thead>
<tr>
<th></th>
<th>R byte-code Interpreter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Instructions</td>
<td>327 M</td>
</tr>
<tr>
<td>SEXPREC Object Allocated</td>
<td>20</td>
</tr>
<tr>
<td>VECTOR Scalar Allocated</td>
<td>1 M</td>
</tr>
<tr>
<td>VECTOR Non-scalar Allocated</td>
<td>2</td>
</tr>
</tbody>
</table>
Related Work

Program Types

- Type I (Loop)
- Type II (Vector)
- Type III (Library)

Program Types

- ORBIT
- pqR
- R Byte-code Interpreter
- Revolution R
- TERR

Legend

- No JIT
- JIT to native code
- Interpreter level JIT

Our work

Non-compatible

- LLVM R
- Rapydo (PyPy)
- TruffleR (Java)
- FastR (Java)
- Riposte
- Renjin (Java)
The GNU R VM

- **Default Interpreter**
  - AST interpreter
- **Byte-code Interpreter**
  - Stack VM based interpreter
- **Both interpreters**
  - Share the same R runtime environment
  - Use the same object model
Problems Analysis – Slow Speed

- Reasons
  - Common problems of Dynamic scripting languages
    • ...
  - R specific semantics
    • ...
  - Overhead from R’s generic object representation
    • Instructions for allocation and garbage collection
Problem Analysis - Memory Consumption/Allocation

- Generic Object Representation
  - Two basic meta object types for all

  - All runtime and user type objects are expressed with the two types
    - E.g. local frame context: linked list by node objects
    - E.g. matrix: vector object (data) + linked list(attributes) + vector objects (‘dim’, dim sizes)
Optimizations in GNU R

- Improving Speed
  - Translate into byte-code
  - Byte-code interpreter: direct threading code dispatch
  - Classic compiler optimizations to the byte-code
  - Copy-on-write
  - ...

- Optimizing Memory system
  - Memory Allocator
    - Pre-allocate pages of SEXPREC
    - Pre-allocates different sizes of small VECTOR_SEXPRECV
  - Garbage Collator
    - Stop-world, multi-generation based collector

The generic byte-code instruction set does not change!
The representation of generic objects does not change!
**ORBIT — Optimized R Byte-code Interpreter**

- **Focus on Specialization**
  - Generic byte-code → type specialized byte-code
  - Generic data representation → specialized data representation

- **Rely on runtime feedback**
  - Aggressive: profile once → speculative typing

- Pure interpreter approach, no native code generation

- Be compatible with the GNU R implementation

---

**Legend**

<table>
<thead>
<tr>
<th>Original Component</th>
<th>New Component</th>
</tr>
</thead>
</table>

---

**Diagram:**

- R Byte Code Compiler → Byte-code → R Opt Byte-Code Compiler
- Specialized byte-code
- Code Selection and Guard Failure Roll Back
- Specialized Byte-Code Execution Extension
- Runtime Profiling Component
- R Byte-code Interpreter

---

11
Optimizing R VM: Allocation Removal and Path Length Reduction via Interpreter-level Specialization

An Example of ORBIT Specialization

Source

```r
foo <- function(a) {
  b <- a + 1
}
```

Byte-code Symbol table

<table>
<thead>
<tr>
<th>Idx</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;a&quot;</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>a+1</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
</tr>
</tbody>
</table>

Generic Domain

<table>
<thead>
<tr>
<th>PC</th>
<th>STMTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GETVAR, 1</td>
</tr>
<tr>
<td>3</td>
<td>LDCONST, 2</td>
</tr>
<tr>
<td>5</td>
<td>ADD, 3</td>
</tr>
<tr>
<td>7</td>
<td>SETVAR, 4</td>
</tr>
<tr>
<td>9</td>
<td>INVISIBLE</td>
</tr>
<tr>
<td>10</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

Specialized Domain

```

ORBIT

```

Profile point

Original data representation

VM Stack

SEXPREC ptr
SEXPREC ptr
SEXPREC ptr

1
VECTOR

1
VECTOR

SEXPREC ptr

Specialized data representation

VM Stack

real scalar
real scalar
SEXPREC ptr
ORBIT Approach Highlight

- **Type profiling + Fast type inference**
  - Profiling once -> trigger optimization
  - Simple type system, use profiling type to help typing

- **Specialized data representation**
  - Use raw (unboxed) objects to replace generic objects
  - Mixed Stack to store boxed and unboxed objects
  - With a type stack to track unboxed objects in the stack
  - Unbox value cache: a software cache for faster local frame object access

- **Specialized byte-code and runtime function routines**
  - Type specialized instructions for common operations
  - Simplify calling conventions according to R’s semantics

- **Guards to handle incorrect type speculation**
  - Type change → Guard failure → Restore the generic code and object
  - Combine the new type with the original profiling type → Retry optimization later
Optimizing R VM: Allocation Removal and Path Length Reduction via Interpreter-level Specialization

For Loop Performance Metrics

```r
r <- 0;
for( i in 1:1000000) {
  r <- r + i;
}
print(r);
```

<table>
<thead>
<tr>
<th></th>
<th>R byte-code</th>
<th>ORBIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Instructions</td>
<td>327 M</td>
<td>98 M</td>
</tr>
<tr>
<td>SEXPREC Object Allocated</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>VECTOR Scalar Allocated</td>
<td>1 M</td>
<td>9</td>
</tr>
<tr>
<td>VECTOR Non-scalar Allocated</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

- Memory allocated removed
  - The long 1:1000000 object
  - New “r” value used in each iteration
Performance Evaluation

- **Benchmarks – Type I code**
  - Scalar benchmark suite

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRT</td>
<td>Chinese Remainder Theorem</td>
</tr>
<tr>
<td>Fib</td>
<td>fibonacci number, iterative method</td>
</tr>
<tr>
<td>Sum</td>
<td>For loop based accumulation</td>
</tr>
<tr>
<td>GCD</td>
<td>Greatest Common Divisor for 100M pairs of random numbers</td>
</tr>
<tr>
<td>Primes</td>
<td>Find prime numbers</td>
</tr>
</tbody>
</table>

- Shootout benchmark suite
  - nbbody, fannkuch-redux, spectral-norm, mandelbrot, pidigits

- **Environment**
  - CPU: Xeon(R) CPU E31245 @3.30GHz (Turbo boost disabled)
  - Linux: Fedora 16 (3.1.0-0.rc10.git0.1.fc16.x86_64)
  - R VMs:
    - Byte-code interpreter: R-2.14.1 with byte-code compiling enabled
    - ORBIT: R-2.14.1 with ORBIT extensions

CRT: Chinese Remainder Theorem
Fib: fibonacci number, iterative method
Sum: For loop based accumulation
GCD: Greatest Common Divisor for 100M pairs of random numbers
Primes: Find prime numbers
Performance of ORBIT – Scalar Benchmark

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SEXPREC</th>
<th>VECTOR scalar</th>
<th>VECTOR non-scalar</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRT</td>
<td>76.06%</td>
<td>82.83%</td>
<td>97.58%</td>
</tr>
<tr>
<td>Fib</td>
<td>99.16%</td>
<td>99.99%</td>
<td>100%</td>
</tr>
<tr>
<td>Primes</td>
<td>98.21%</td>
<td>94.70%</td>
<td>50.00%</td>
</tr>
<tr>
<td>Sum</td>
<td>15.00%</td>
<td>99.99%</td>
<td>100%</td>
</tr>
<tr>
<td>GCD</td>
<td>99.99%</td>
<td>99.99%</td>
<td>25.00%</td>
</tr>
<tr>
<td>Mean</td>
<td>77.68%</td>
<td>95.50%</td>
<td>74.52%</td>
</tr>
</tbody>
</table>
Performance of ORBIT – Shootout Benchmark

Percentage of Memory Allocation Reduced

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SEXPREC</th>
<th>VECTOR scalar</th>
<th>VECTOR non-scalar</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbody</td>
<td>85.47%</td>
<td>86.82%</td>
<td>69.02%</td>
</tr>
<tr>
<td>fannkuch-redux</td>
<td>99.99%</td>
<td>99.30%</td>
<td>71.98%</td>
</tr>
<tr>
<td>spectral-norm</td>
<td>43.05%</td>
<td>91.46%</td>
<td>99.46%</td>
</tr>
<tr>
<td>mandelbrot</td>
<td>99.95%</td>
<td>99.99%</td>
<td>99.99%</td>
</tr>
<tr>
<td>pidigits</td>
<td>96.89%</td>
<td>98.37%</td>
<td>95.13%</td>
</tr>
<tr>
<td>Binary-trees</td>
<td>36.32%</td>
<td>67.14%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Mean</td>
<td>76.95%</td>
<td>90.51%</td>
<td>72.60%</td>
</tr>
</tbody>
</table>

Speedup over byte-code interpreter

Dominated by user level call overhead. Not handled by ORBIT
Conclusion

- **Our Work**
  - Revealed Generic Object Representation is a key source of low performance
  - Focused on specialization
    - Operation specialization + Object representation specialization
  - Implemented a JIT engine, pure interpreter based
  - Reduced instruction path length and memory allocations

- **Next Step**

  **Shootout – Slowdown to C implementation**

  - **205 x** R Byte-code
  - **56 x** ORBIT

- **Need Better Benchmarks for R**
  - An community effort: https://github.com/rbenchmark/benchmarks
Thank You!

Contact Info:
Haichuan Wang (hwang154@illinois.edu)
Peng Wu (pengwu@acm.org)
David Padua (padua@illinois.edu)