Directions in Statistical Computing
2014
Renjin's JIT

Thinking about R as a Query Language

Alexander Bertram
BeDataDriven
Quick Intro: Renjin

- R-language Interpreter written in Java, uses GNU R core packages (base, stats, etc) as-is
- Goals: Completeness first, performance next
- C/Fortran: Supported with translator and emulation layer
- Can run roughly ~50% of CRAN packages (see packages.renjin.org)
- Actively user group, diverse
R as a “Query Language”

How can R be as fast as Fortran or C++?

How can R be more like SQL?

- Analyst describes the what
- Query planner determines the how
  - Implicit parallelism
  - Target diverse architecture (in-memory, single node, clusters)
Is R dynamic?

Argument: Not where/when performance matters
But R is too dynamic!

```r
airlines <- read.bigtable("airlines")
print(nrow(airlines)) # ~240m

fit.exp <- function(x, max.iter = 10) {
  rate <- 1 / mean(x)
  repeat {
    loglik <- sum(-dexp(r = rate, x = lambda, log = T)
    if (goodEnough(loglik)) break
    rate <- next
  }
}
```

Complicated Argument Matching

sum() is group generic, dispatches based on argument

Is the break() function redefined?
airlines <- read.bigtable("airlines")
delay <- airlines$delay[airlines$delay > 30]

dexp <- function(x, rate=1, log = FALSE) {
    mean <- 1/rate
    d <- exp(-x / mean) / mean
    if(log) return(log(d))
    d
}

fit.exp <- function(x, max.iter = 10) {
    rate <- 1 / mean(x)
    repeat {
        loglik <- sum(-dexp(r = rate, x, log = T)
        if( logLik > epsilon ) break
        rate <- update(rate)
    }
}

rate <- fit.exp
Real world example: Distance Correlation
[ see energy package]
function (x, y, index = 1) {
  x <- dist(x)
  y <- dist(y)
  x <- as.matrix(x)
  y <- as.matrix(y)
  n <- nrow(x)
  m <- nrow(y)
  dims <- c(n, ncol(x), ncol(y))
  Akl <- function(x) {
    d <- as.matrix(x)^index
    m <- rowMeans(d)
    M <- mean(d)
    a <- sweep(d, 1, m)
    b <- sweep(a, 2, m)
    return(b + M)
  }
  A <- Akl(x)
  B <- Akl(y)
  dCov <- sqrt(mean(A * B))
  dVarX <- sqrt(mean(A * A))
  dVarY <- sqrt(mean(B * B))
  V <- sqrt(dVarX * dVarY)
  if (V > 0)
    dCor <- dCov/V
  else dCor <- 0
  return(list(dCov = dCov, dCor = dCor, dVarX = dVarX, dVarY = dVarY))
}
Optimizations: Views

x <- dist(x)
y <- dist(y)
x <- as.matrix(x)
y <- as.matrix(y)

# GNU R: x^2 + y^2 memory alloc'd
# Renjin: ~ 0
public class DistanceMatrix extends DoubleVector {
    private Vector vector;

    public double getElementAsDouble(int index) {
        int size = vector.length();
        int row = index % size;
        int col = index / size;
        if(row == col) {
            return 0;
        } else {
            double x = vector.getElementAsDouble(row);
            double y = vector.getElementAsDouble(col);
            return Math.abs(x - y);
        }
    }
    
    public int length() { return vector.length() * vector.length(); }
}
Deferred Evaluation

• Defer computation of pure functions when inputs exceed some threshold:

```r
x <- (1:100) + 4  # x is computed
y <- (1:e^6) + 4   # no work done
                 # x is a view
z <- y - mean(z)
z <- dnorm(z)
print(z)  # triggers evaluation
```
function (x, y, index = 1) {
    x <- dist(x)
    y <- dist(y)
    x <- as.matrix(x)
    y <- as.matrix(y)
    n <- nrow(x)
    m <- nrow(y)
    dims <- c(n, ncol(x), ncol(y))
    Akl1 <- function(x) {
        d <- as.matrix(x)^index
        m <- rowMeans(d)
        M <- mean(d)
        a <- sweep(d, 1, m)
        b <- sweep(a, 2, m)
        return(b + M)
    }
    A <- Akl1(x)
    B <- Akl1(y)
    dCov <- sqrt(mean(A * B))
    dVarX <- sqrt(mean(A * A))
    dVarY <- sqrt(mean(B * B))
    V <- sqrt(dVarX * dVarY)
    if (V > 0)
        dCor <- dCov/V
    else dCor <- 0
    return(list(dCov = dCov, dCor = dCor)}
Query Planner

- Once evaluation is triggered: we have a better broad view of the calculation to be completed
- Computation Graph is essentially a pure function
- We can reorder operations, and easily see which branches can be evaluated independently, in parallel
function (x, y, index = 1) {
    x <- dist(x)
    y <- dist(y)
    x <- as.matrix(x)
    y <- as.matrix(y)
    n <- nrow(x)
    m <- nrow(y)
    dims <- c(n, ncol(x), ncol(y))
    Akl <- function(x) {
        d <- as.matrix(x)^index
        m <- rowMeans(d)
        M <- mean(d)
        a <- sweep(d, 1, m)
        b <- sweep(a, 2, m)
        return(b + M)
    }
    A <- Akl(x)
    B <- Akl(y)
    dCov <- sqrt(mean(A * B))
    dVarX <- sqrt(mean(A * A))
    dVarY <- sqrt(mean(B * B))
    V <- sqrt(dVarX * dVarY)
    if (V > 0) {
        dCor <- dCov/V
    } else {
        dCor <- 0
    }
    return(list(dCov = dCov, dCor = dCor)
}
Loop Fusion

\[ \text{mean}(\text{op1}(\text{op2}(\text{op3}(x)))) \]

**transformed to...**

```java
double sum = 0;
for(int i..1000) {
    sum += op1(op2(op3))
}
```
Beyond Bytecode

JVM Byte Code → Native Machine Code

SQL Query

OpenCL
Results
**Issue 61: Simple function won't work**

1 person starred this issue and may be notified of changes.

**Status: New**

**Owner: ****

Reported by radfordneal, Jul 23, 2013

Hi. I tried the following simple function both on your on-line demo and on a downloaded version on Windows 7, with Java

```r
f <- function () { b <- 0; a <- rep(1.1,1000); for (i in 1:100000) { a <- sqrt(a+7); b <- b + sum(a); } b
```

When I call it with `f()`, it just hangs or eventually gives an error. However, it works when the `rep` by 1000 is replaced rep by 100.
Loops!

```r
m <- 4
for (i in 1:m) {
  x = exp(tanh(a^2 * (b^2 + i/m)))
  r[i%%10+1] = r[i%%10+1] + sum(x)
}
```

Kaboom!
(thanks Radford!)
Loops!

- R gives you the flexibility to mix imperative with functional approaches
- In many dynamic languages (JS, Ruby), sophisticated runtime analysis is required to identify and compile hotspots in the code.
- In R, they're pretty easy to spot:
  ```r
  x <- 1:1e6
  for(i in seq_along(x)) {
    ...
  }
  ```
for (i in 1:m) {
    x = exp (tanh (a^2 * (b^2 + i/m)))
    r[i%%10+1] = r[i%%10+1] + sum(x)
}

BB1:
τ₃ ← (: 1.0d m₀)
Λ₀₁ ← 0
τ₂ ← length(τ₃)

BB2: [L0]
r₁ ← Φ(r₀, r₂)
Λ₀₂ ← Φ(Λ₀₁, Λ₀₃)
i₁ ← Φ(i₀, i₂)
χ₁ ← Φ(x₀, x₂)
if Λ₀₂ >= τ₂ => TRUE:L3,
FALSE:L1, NA:ERROR

BB3: [L1]
i₂ ← τ₃[Λ₀₂]
τ₄ ← (^ a₀ 2.0d)
τ₅ ← (^ b₀ 2.0d)
τ₆ ← (/ i₂ m₀)
τ₇ ← (+ τ₅ τ₆)
τ₈ ← (* τ₄ τ₇)
τ₉ ← (tanh τ₈)
τ₁₀ ← (exp τ₉)
τ₁₁ ← (+ τ₁₀ 1.0d)
τ₁₂ ← ([ r₁ τ₁₁)
τ₁₃ ← (sum x₂)
τ₁₄ ← (i₂ 10.0d)
τ₁₅ ← (+ τ₁₄ 1.0d)
τ₁₆ ← (i₂ 10.0d)
τ₁₇ ← (+ τ₁₆ 1.0d)
r₂ ← ([<- r₁ τ₁₇)

BB4: [L2]
Λ₀₃ ← increment counter Λ₀₂
goto L0

BB5: [L3]
return NULL
Compared to other dynamic languages?

- **Argument:** Speculative specialization works very well for long-running code, but unnecessary for most statistical code with many loops:
  - Simulations
  - Iterative algorithms
  - ?

- **Needs to be tested...**
# Packages

<table>
<thead>
<tr>
<th>Package</th>
<th>Downstream</th>
<th>Languages</th>
<th>Problems</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3</td>
<td></td>
<td></td>
<td>TF</td>
<td>A3: Accurate, Adaptable, and Accessible Error Metrics for Predictive Models</td>
</tr>
<tr>
<td>abc</td>
<td></td>
<td></td>
<td></td>
<td>Tools for Approximate Bayesian Computation (ABC)</td>
</tr>
<tr>
<td>abcdeFBA</td>
<td></td>
<td></td>
<td>TF</td>
<td>ABCDE_FBA: A-Biologist-Can-Do-Everything of Flux Balance Analysis with this package.</td>
</tr>
<tr>
<td>ABCExtremes</td>
<td></td>
<td></td>
<td>TF</td>
<td>ABC Extremes</td>
</tr>
<tr>
<td>ABCp2</td>
<td></td>
<td></td>
<td>TF</td>
<td>Approximate Bayesian Computational model for estimating P2</td>
</tr>
<tr>
<td>abctools</td>
<td></td>
<td>C</td>
<td></td>
<td>Tools for ABC analyses</td>
</tr>
<tr>
<td>abd</td>
<td></td>
<td></td>
<td></td>
<td>The Analysis of Biological Data</td>
</tr>
<tr>
<td>abind</td>
<td></td>
<td></td>
<td>TF</td>
<td>Combine multi-dimensional arrays</td>
</tr>
<tr>
<td>aBioMarVsuit</td>
<td></td>
<td></td>
<td></td>
<td>A Biomarker Validation Suit for predicting Survival using gene signature.</td>
</tr>
<tr>
<td>abn</td>
<td></td>
<td>C</td>
<td></td>
<td>Data Modelling with Additive Bayesian Networks</td>
</tr>
<tr>
<td>AcceptanceSampling</td>
<td></td>
<td></td>
<td></td>
<td>Creation and evaluation of Acceptance Sampling Plans</td>
</tr>
<tr>
<td>ACCLMA</td>
<td></td>
<td></td>
<td>TF</td>
<td>ACC &amp; LMA Graph Plotting</td>
</tr>
<tr>
<td>ACD</td>
<td></td>
<td></td>
<td>TF</td>
<td>Categorical data analysis with complete or missing responses</td>
</tr>
<tr>
<td>Ace</td>
<td></td>
<td></td>
<td>TF</td>
<td>Assay-based Cross-sectional Estimation of incidence rates</td>
</tr>
<tr>
<td>acepack</td>
<td></td>
<td>Fortran</td>
<td>TF</td>
<td>ace() and avas() for selecting regression transformations</td>
</tr>
<tr>
<td>acer</td>
<td></td>
<td>C++</td>
<td></td>
<td>The ACER Method for Extreme Value Estimation</td>
</tr>
<tr>
<td>aCGH.Spline</td>
<td></td>
<td></td>
<td></td>
<td>Robust spline interpolation for dual color array comparative genomic hybridization data</td>
</tr>
<tr>
<td>ACNE</td>
<td></td>
<td></td>
<td></td>
<td>Affymetrix SNP probe-summation using non-negative matrix factorization</td>
</tr>
<tr>
<td>aCRM</td>
<td></td>
<td></td>
<td>TF</td>
<td>Convenience functions for analytical Customer Relationship Management</td>
</tr>
<tr>
<td>acs</td>
<td></td>
<td></td>
<td></td>
<td>Download and manipulate data from the US Census American Community Survey</td>
</tr>
<tr>
<td>Actigraphy</td>
<td></td>
<td></td>
<td></td>
<td>Actigraphy Data Analysis</td>
</tr>
<tr>
<td>actuariar</td>
<td></td>
<td>C</td>
<td>TF</td>
<td>Actuarial functions</td>
</tr>
</tbody>
</table>
Developing CI + benchmarking system for testing optimizations
More Information

- http://www.renjin.org
- http://packages.renjin.org