UT Southwestern
Medical Center

Optimization in

Noah Chang
Tao Wang Lab - Lab Meeting
June 28, 2024

Good practices

Basic Software Engineering Principles

```
some_dataframe[some_column == some_value, ]
```

Time Complexity?

```
some_list["some_key"]
```

Time Complexity?

Basic Software Engineering Principles

```
some_dataframe[some_column == some_value, ]
```

Time Complexity? O(n)

```
some_list["some_key"]
```

Time Complexity?

Basic Software Engineering Principles

```
some_dataframe[some_column == some_value, ]
```

Time Complexity? O(n)

```
some_list["some_key"]
```

Time Complexity? O(1)

Using hash map data structure can be faster than your usual data.frame

Vectorized Operation in R

```r
# Inefficient
result <- numeric(1000)
for (i in 1:1000) {
  result[i] <- i + 1
}

# Efficient
result <- 1:1000 + 1
```

```r
# Inefficient
result <- numeric(10)
for (i in 1:10) {
  result[i] <- sum(1:i)
}

# Efficient
result <- sapply(1:10, function(x) sum(1:x))
```

Memory Re-allocation

```r
# Memory Reallocation 1: Growing a vector in a loop
n <- 10000
vec <- numeric(0)  # Start with an empty vector
for (i in 1:n) {
  vec <- c(vec, i)  # Append the current value of i to the vector
}

#  Memory Reallocation 2: Growing a data.frame in a loop
df <- data.frame()
for (i in 1:n) {
  new_row <- data.frame(a = i, b = rnorm(1))
  df <- rbind(df, new_row)  # Append the new row to the data frame
}
```

Memory Re-allocation - Solution

```
# Efficient way 1: Preallocating the vector
n <- 10000
vec <- numeric(n)  # Preallocate a vector of length n
for (i in 1:n) {
  vec[i] <- i  # Assign the value of i directly to the preallocated vector
}

# Efficient way 2: Preallocating the data frame
df <- data.frame(a = numeric(n), b = numeric(n))  # Preallocate a df of length n
for (i in 1:n) {
  df$a[i] <- i
  df$b[i] <- rnorm(1)
}
```

Fast packages for optimizations

Data Manipulation package comparisons

Feature/Aspect	Base data.frame	dplyr	data.table
Syntax Simplicity	Traditional, less readable	Intuitive, chainable verbs (%>%)	Concise, uses DT[i, j, by] syntax
Performance	Moderate	Moderate to High (depends on backend)	High
Data Manipulation	Uses base R functions	Uses a suite of verbs (filter, select, mutate, etc.)	Uses in-place updates with :=, optimized for speed
Memory Efficiency	Copies data frequently	Copies data in some operations	Modifies data by reference
Grouping and Aggregation	Uses tapply, aggregate, by	Uses group_by and summarise	Uses by argument and optimized jexpression
Learning Curve	Moderate	Easy (especially for those familiar with SQL)	Steeper than dplyr, but powerful
Handling Large Data	Less efficient	More efficient with dplyr backends like data.table or dtplyr	Very efficient
Integration with Other Packages	High (standard base R)	High (tidyverseecosystem)	High, especially with data manipulation packages like dplyr
Complex Operations	Can be verbose and complex	Simplified with chaining and functions	Highly efficient but requires knowledge of syntax
Join Operations	Uses merge	Uses left_join, inner_join, etc.	Uses merge with optimized performance

Data Manipulation package comparisons

```r
# Base Data.Frame
df <- data.frame(group = rep(c("A", "B", "C"), each = 10),
                 value = rnorm(30))

# Using aggregate function in base R
mean_df <- aggregate(value ~ group, data = df, FUN = mean)

# Using dplyr
mean_df <- df %>%
  group_by(group) %>%
  summarise(mean_value = mean(value, na.rm = TRUE))

# Using data.table
dt <- as.data.table(df)
mean_dt <- dt[, .(mean_value = mean(value, na.rm = TRUE)), by = group]
```

Package comparisons - groupby

Input table: 1,000,000,000 rows x 9 columns (50 GB)

Package	Version	Date	Time
Polars	0.8.8	2021-06-30	143s
data.table	1.14.1	2021-06-30	155s
DataFrames.jl	1.1.1	2021-05-15	200s
ClickHouse	21.3.2.5	2021-05-12	256s
cuDF*	0.19.2	2021-05-31	492s
spark	3.1.2	2021-05-31	568s
(py)datatable	1.0.0a0	2021-06-30	730s
dplyr	1.0.7	2021-06-20	internal error
pandas	1.2.5	2021-06-30	out of memory
dask	2021.04.1	2021-05-09	out of memory
Arrow	4.0.1	2021-05-31	internal error
DuckDB*	0.2.7	2021-06-15	out of memory
Modin		see README	pending

https://h2oai.github.io/db-benchmark/

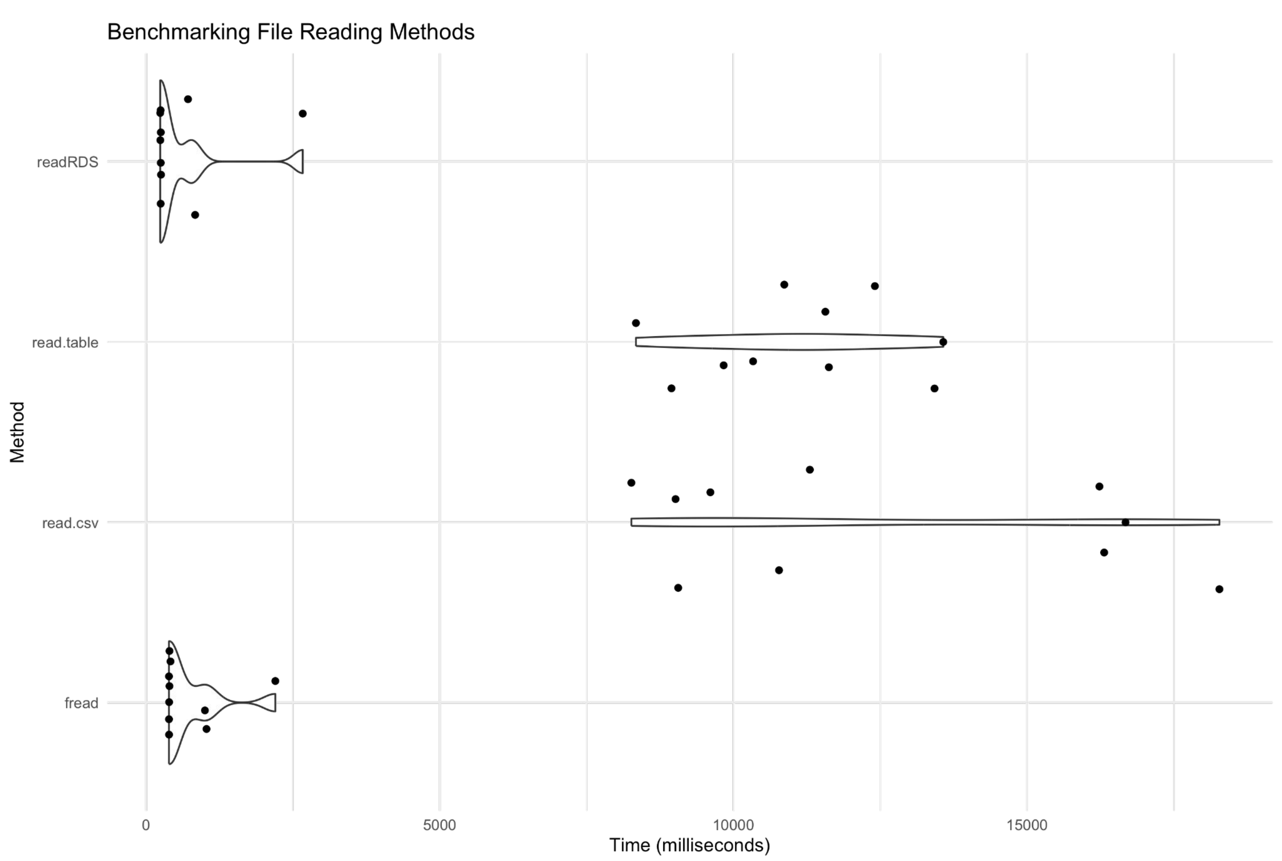
Package comparisons - join

Input table: 100,000,000 rows x 7 columns (5 GB)

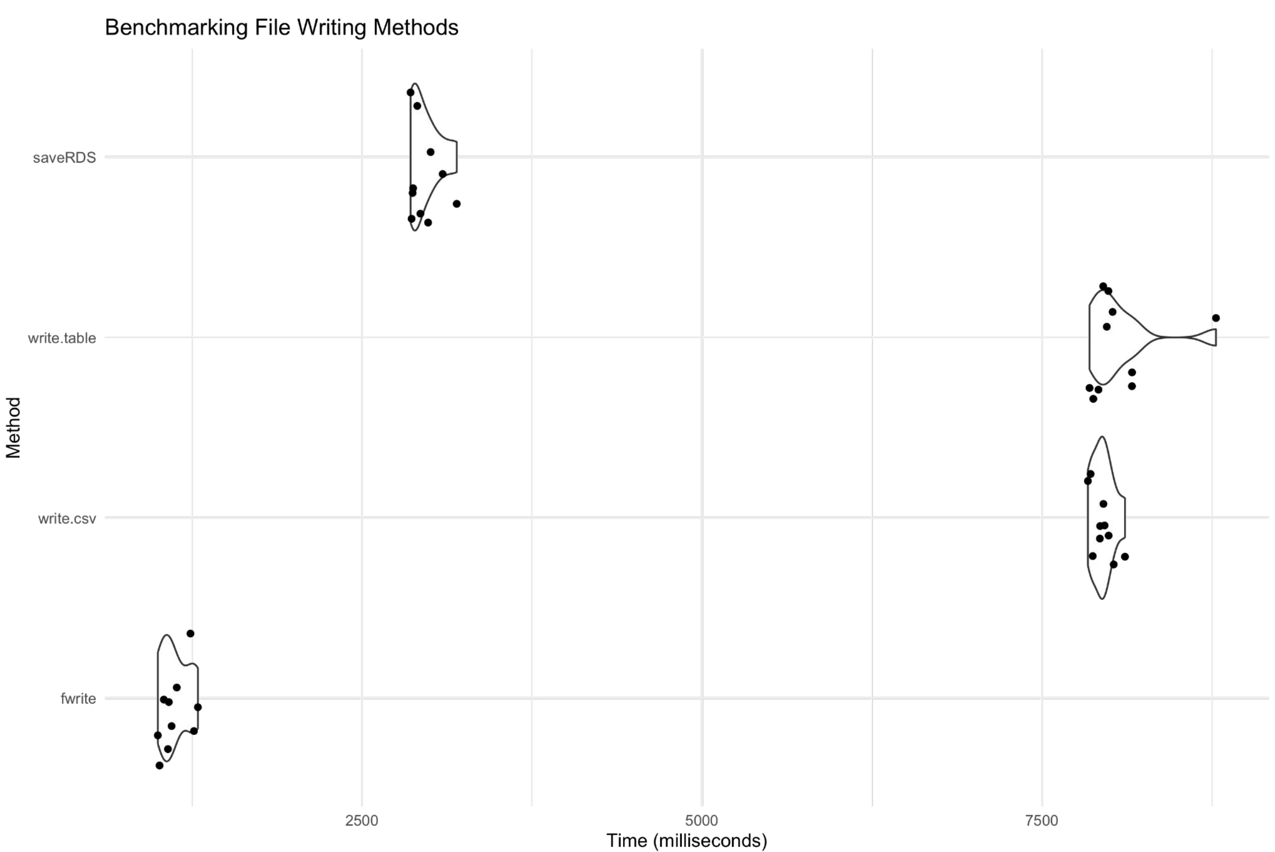
Package	Version	Date	Time
Polars	0.8.8	2021-06-30	43s
data.table	1.14.1	2021-06-30	92s
ClickHouse	21.3.2.5	2021-05-12	159s
spark	3.1.2	2021-05-31	332s
DataFrames.jl	1.1.1	2021-06-03	349s
dplyr	1.0.7	2021-06-20	370s
(py)datatable	1.0.0a0	2021-06-30	500s
pandas	1.2.5	2021-06-30	628s
DuckDB	0.2.7	2021-06-15	630s
dask	2021.04.1	2021-05-09	internal error
cuDF*	0.19.2	2021-05-31	internal error
Arrow	4.0.1	2021-05-31	not yet implemented
Modin		see README	pending

https://h2oai.github.io/db-benchmark/

Package comparisons - reading files



Package comparisons - writing files



Binding Rows - Speed comparison

https://rpubs.com/jimhester/rbind

C++ for optimizations

Rcpp

What is Rcpp?

Rcpp is an R package that facilitates the seamless integration of R and C++ code. It allows R users to write high-performance C++ code and call it directly from R, thereby combining the ease of R with the speed of C++.

Why Use Rcpp?

- **Performance:** C++ is significantly faster than R for many operations, especially those involving loops or complex computations.
- **Flexibility:** C++ allows for more control over memory management and optimization.
- **Integration:** Rcpp provides a smooth interface between R and C++, making it easy to pass data back and forth.

Rcpp - Usage 1: cppFunction()

```
library("Rcpp")
cppFunction('
  double add_cpp(double x, double y) {
    double value = x + y;
    return value;
  }
')
add_cpp(1, 2)
#> [1] 3
```

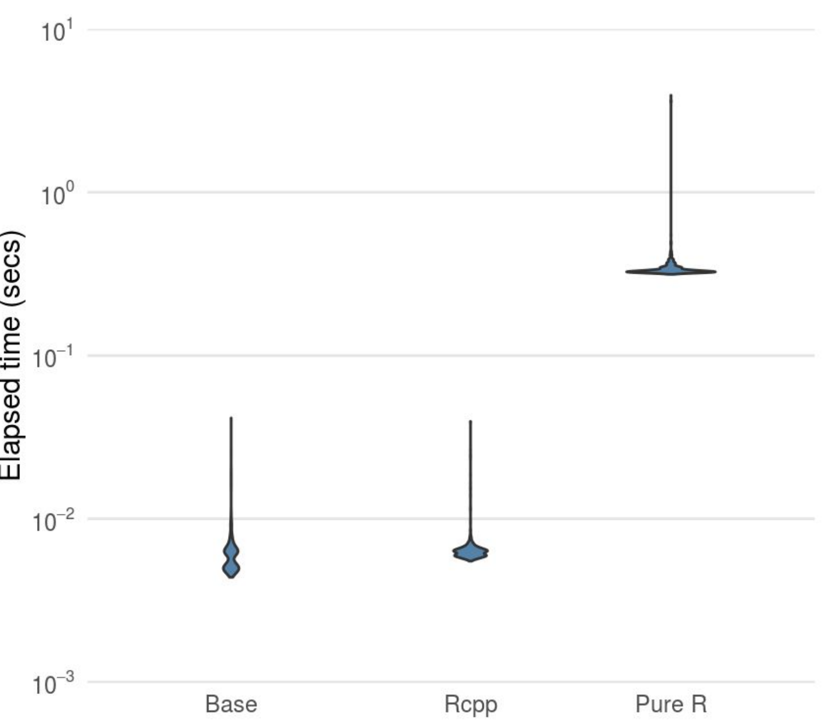
Rcpp - Usage 2: sourceCpp()

```cpp
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
double add_cpp(double x, double y) {
  double value = x + y;
  return value;
}
```

```r
# In R
library("Rcpp")
sourceCpp("path/to/file.cpp")
add_cpp(1, 2)
#> [1] 3
```

Rcpp - Performance Comparison



https://bookdown.org/csgillespie/efficientR/performance.html

RcppArmadillo

What is RcppArmadillo?

RcppArmadillo is an R package that provides a seamless interface between R and Armadillo, a high-performance C++ linear algebra library. It combines the ease of Rcpp with the speed and flexibility of Armadillo, making it ideal for tasks involving complex linear algebra computations.

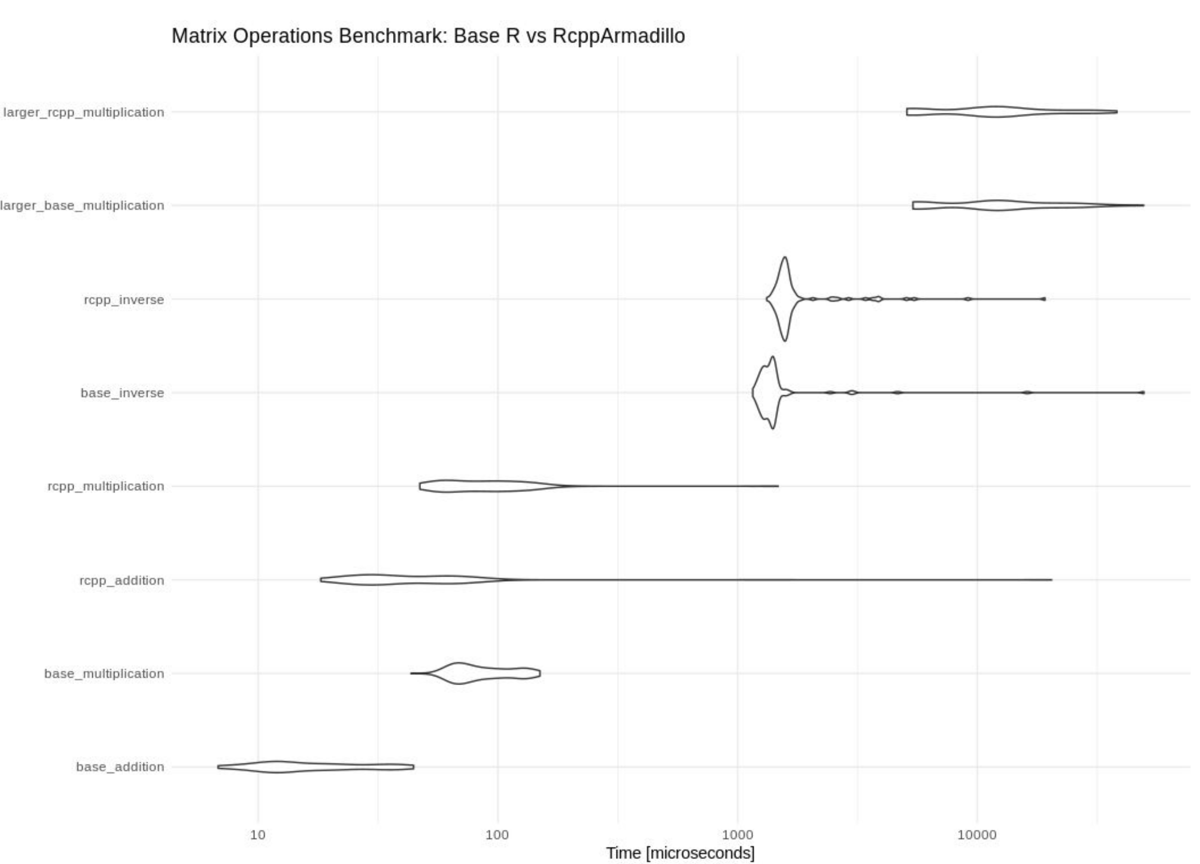
Why Use RcppArmadillo?

- **Performance:** Armadillo offers highly optimized linear algebra routines that can outperform equivalent R code.
- **Ease of Use:** Armadillo syntax is similar to MATLAB, making it easy to write and read.
- **Integration:** RcppArmadillo provides a smooth integration with R, allowing for efficient data transfer between R and C++.

RcppArmadillo - usage

```
# Define RcppArmadillo functions
cppFunction(depends = "RcppArmadillo", code = '
arma::mat addMatrices(const arma::mat& A, const arma::mat& B) {
    return A + B;
}
')
cppFunction(depends = "RcppArmadillo", code = '
arma::mat multiplyMatrices(const arma::mat& A, const arma::mat& B) {
    return A * B;
}
')
cppFunction(depends = "RcppArmadillo", code = '
arma::mat invertMatrix(const arma::mat& A) {
    return arma::inv(A);
}
')
```

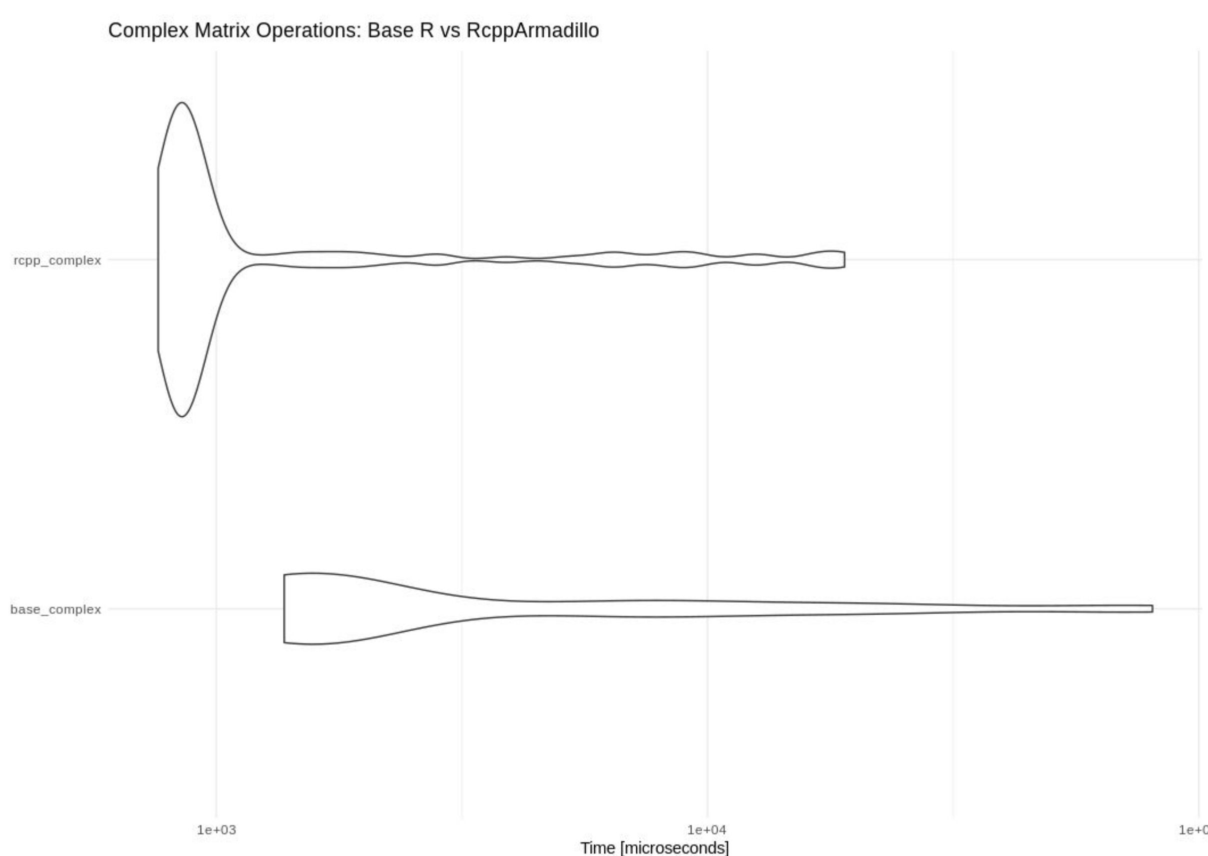
RcppArmadillo - Basic operations benchmark



RcppArmadillo - usage

```
# Complex series of matrix operations
cppFunction(depends = "RcppArmadillo", code = '
arma::mat complexOperations(const arma::mat& A, const arma::mat& B) {
    arma::mat C = A + B;                        // Addition
    arma::mat D = A * B;                        // Multiplication
    arma::mat E = arma::inv(C);                 // Inversion
    arma::mat F = D.t();                        // Transposition
    arma::mat G = E % F;                        // Element-wise multiplication
    arma::mat H = arma::chol(G + arma::eye(
      G.n_rows, G.n_cols)
      ); // Cholesky decomposition
    return H;
}
')
```

RcppArmadillo - Complex operations benchmark



Parallel Computing

Future

```r
library(future)
plan(multisession)  # or plan(multicore) on Unix-based systems

result <- future_lapply(1:10, function(x) {
  Sys.sleep(1)
  x^2
})
print(result)
```

OpenMP

```
#include <omp.h>
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::NumericVector parallelVectorAdd(
  Rcpp::NumericVector a,
  Rcpp::NumericVector b
  ) {
  int n = a.size();
  Rcpp::NumericVector result(n);

  #pragma omp parallel for
  for (int i = 0; i < n; ++i) {
    result[i] = a[i] + b[i];
  }
  return result;
}
```

RcppParallel

```cpp
#include <RcppParallel.h>
using namespace RcppParallel;

struct SquareRoot : public Worker {
  const RVector<double> input;
  RVector<double> output;

  SquareRoot(const Rcpp::NumericVector input, Rcpp::NumericVector output)
    : input(input), output(output) {}

  void operator()(std::size_t begin, std::size_t end) {
    for (std::size_t i = begin; i < end; i++) {
      output[i] = sqrt(input[i]);
    }
  }
};

// [[Rcpp::export]]
Rcpp::NumericVector parallelSqrt(Rcpp::NumericVector x) {
  Rcpp::NumericVector output(x.size());
  SquareRoot sqrtWorker(x, output);
  parallelFor(0, x.size(), sqrtWorker);
  return output;
}
```

Parallel method comparisons

Feature	future	OpenMP	RcppParallel
Ease of Use	High	Moderate	Moderate
Backend Flexibility	High (multicore, multisession, etc.)	Low (shared-memory only)	Low (shared-memory only)
Performance	Moderate	High	High
Control	Low (high-level abstraction)	High (fine-grained control)	Moderate (high-level but customizable)
Setup Complexity	Low	High	Moderate
Required Knowledge	Basic R	C++ and OpenMP	C++ TBB
Suitable For	High-level parallelism, distributed computing	Fine-grained, thread-level parallelism	Parallelizing C++ code with Rcpp
RcppArmadillo Compatibility	Not compatible	Requires BLAS compiling with OpenMP support	Compatible out of box

Further Considerations

Documenting and Commenting R Code for better maintenance and update:

- **Descriptive names:** Stop using foo bar temp df
- **Document functions:** Good practice to create a docstring for your functions in roxygen2 format for future R package
- **Create Sections and modularize**: Putting your code into several contained sections and function will be easier to maintain and troubleshoot

Evaluating Efficient R Code:

- **Profiling Tools:** Tools like profvis and Rprof can identify bottlenecks.
- **Memory Management:** data.table over data.frame, and garbage collection.

Thank you for listening

Questions, thoughts, or concerns?