

# R goes Mobile: Efficient Scheduling for Parallel R Programs on Heterogeneous Embedded Systems



**SFB 876** Providing  
Information by Resource-  
Constrained Data Analysis

Helena Kotthaus, Andreas Lang  
Olaf Neugebauer, Peter Marwedel  
03/07/2017



# Parallel Machine Learning Algorithms

## Challenge:

- Find the best algorithm configuration

→ *Vast search space:*

Algorithms +

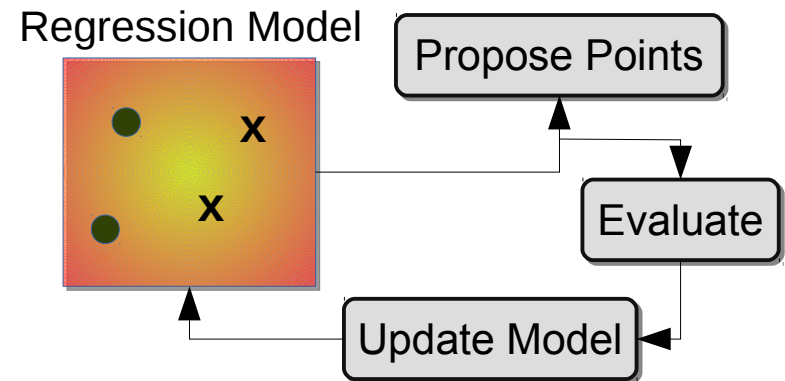
Specific parameters for each

- Parameter tuning can *take weeks*

→ Solution: Reduce evaluations with *model based optimization*

Reduce runtime with efficient *parallel execution*

→ *Enable larger problem sizes*



## Goal:

Resource-aware scheduling strategy for parallel learning algorithms



# R goes Mobile - Parallelizing R on Heterogeneous Architectures

---

## *Challenge:*

- ▶ Running parallel R programs on mobile heterogeneous architectures
  - Tight resources and energy restrictions
  - Parallel execution can cause inefficient utilization
  - Different processors with different frequencies
  - No support

## *Approach:*

- ▶ Enable scheduling of parallel jobs to specific CPUs
- ▶ Use regression model for job runtime estimates
- ▶ Integrate search space exploration and scheduling

## *Goal:*

- ▶ Resource-aware scheduling strategies for parallel R program on embedded devices



[techtimes.com]



# Heterogeneous Architectures

## Odroid XU3 - Used in Mobile Phones

---



### ARM big.LITTLE System

- \* 4 x big - Cortex A15 up to 2.0 GHz
- \* 4 x little - Cortex A7 up to 1.2 GHz

### GPU: Mali-T628

- \* OpenGL ES 3.0/2.0/1.1

### Memory:

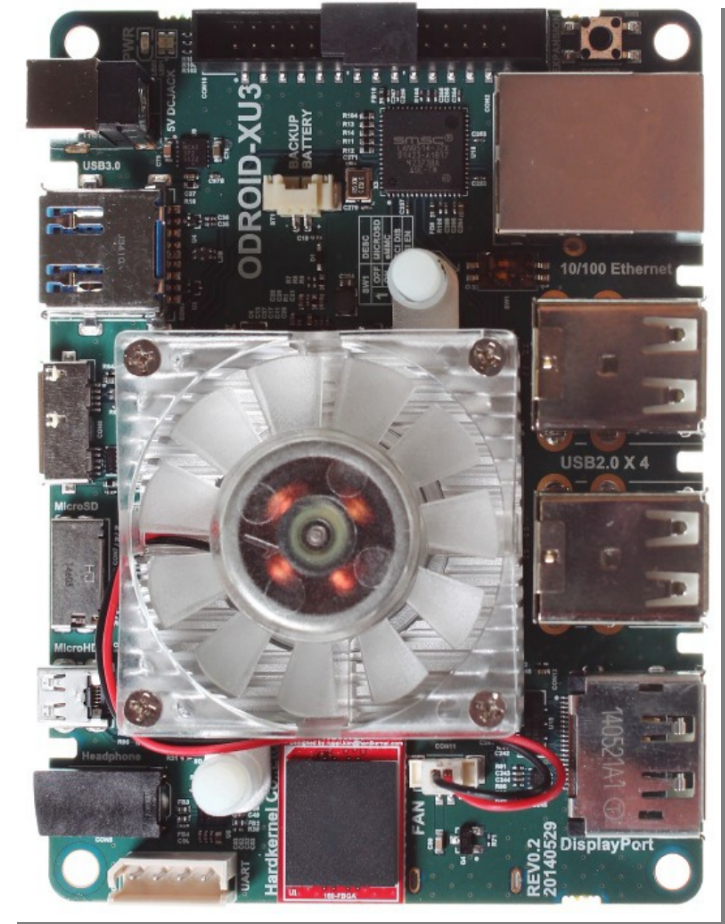
- \* 2GB LPDDR3 RAM

### Power Measurement Sensors:

- \* 4 x TI INA231 (A15, A7, GPU, RAM)

### OS:

- \* Linux and Android



# Allocate Parallel Jobs to specific CPUs

## `mclapply` & `mcpapply`

---

### *mcpapply*

- ▶ Already supports allocation of jobs to specific CPUs with `mc.affinity` (R 3)
- ▶ Disadvantages
  - No controlled execution order
  - Low level

### *mclapply*

- ▶ More convenient
- ▶ But no support for mapping parallel jobs to specific CPUs

### *New hmclapply*

- ▶ Supports mapping to specific CPUs with `cpu.affinity`
- ▶ Controlled scheduling

*How to use `hmclapply` and what about the performance?*



# Allocate Parallel Jobs to specific CPUs

## Exemplary Variance Filter on a Matrix

---

```
devtools::load_all("parallelMap/") # includes modified mclapply
n <- 300 # observations
p <- 20000 # covariates
X <- matrix(replicate(p, rnorm(n, sd = runif(1, 0.1, 10))), n, p)
colnames(X) <- sprintf("var_%05i", 1:p)

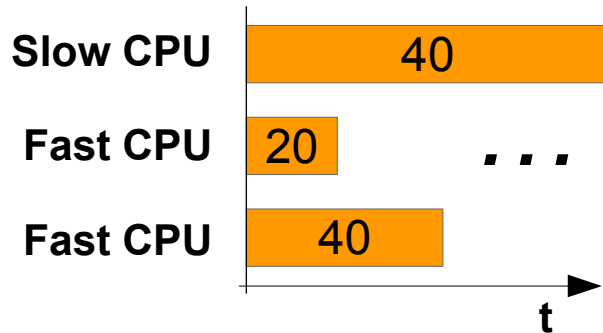
bFunct = function (N){
  for (i in 1:N) {
    train <- sample(nrow(X), 2 / 3 * nrow(X))
    colVars <- apply(X[train, ], 2, var)
    keep <- names(head(sort(colVars, decreasing = TRUE), 100))
    # myAlgorithm(X[, keep])
  }
}

N = c(20,40,40,20,40,40) # different job work loads
affinity = c(4,5,6,4,5,6) # CPU 4 slow, 5 and 6 fast

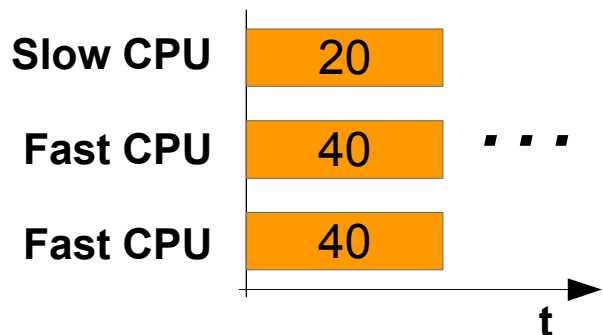
hmclapply(X = N, FUN = bFunct,
          mc.cores = 3, mc.preschedule = FALSE,
          cpu.affinity = affinity)
```

# Results on Heterogeneous Architectures: mclapply vs hmclapply

---



*mclapply* - variance of completion times  
→ 257 (+/- 1.5) seconds



*hmclapply* – balanced times  
→ 234 (+/- 1.0) seconds

→ Efficient job allocation optimizes the overall execution time

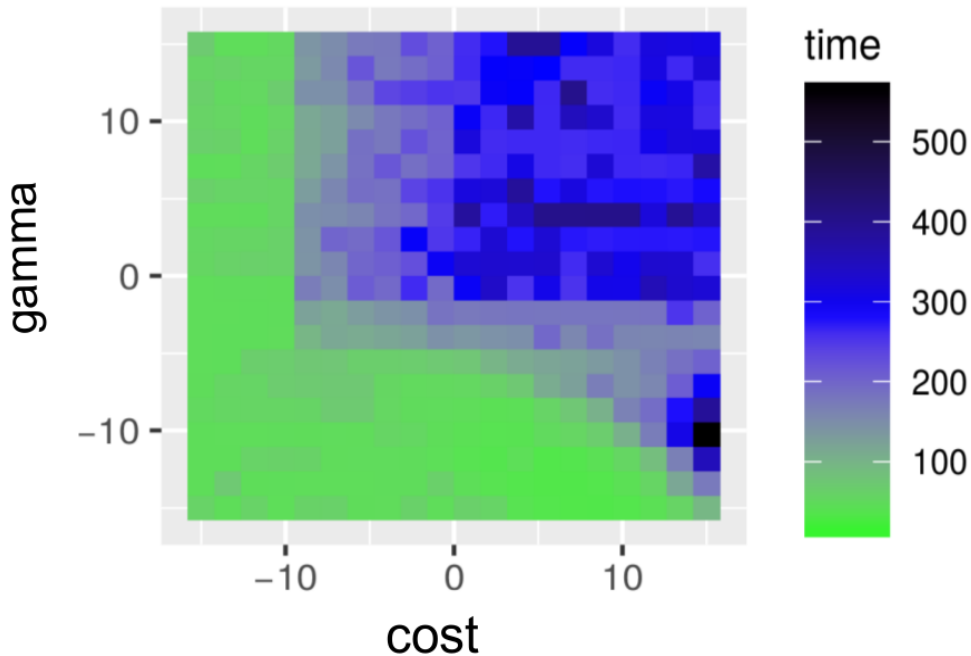
## Problem

→ Efficient scheduling needs to know the runtime of a job for each available processor type

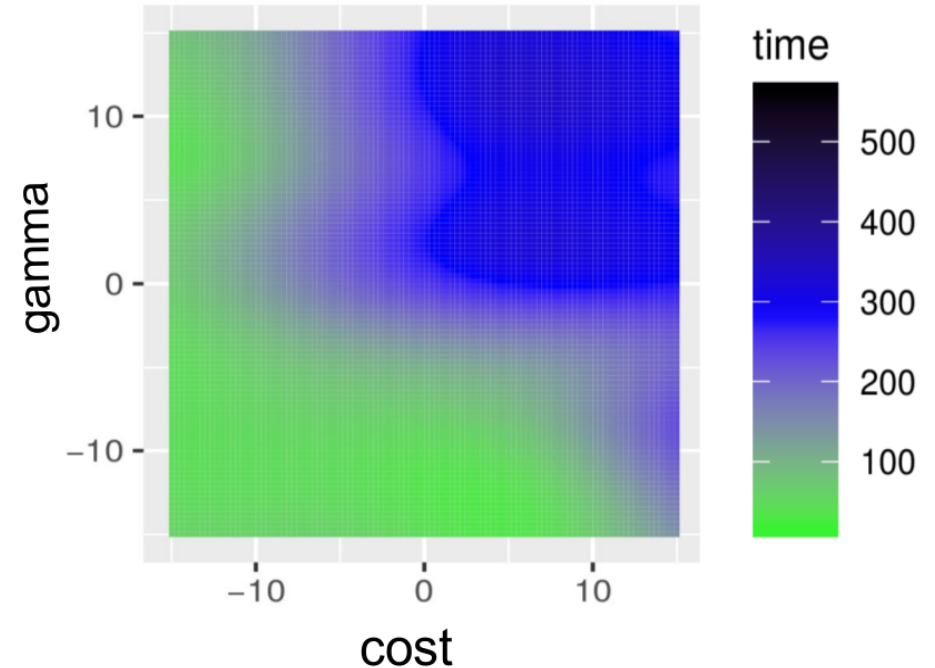
# Solution: Runtime Estimation via Regression Model



## Real Runtime



## Estimated Runtime



→ *Resource estimates are used to guide scheduling strategies*

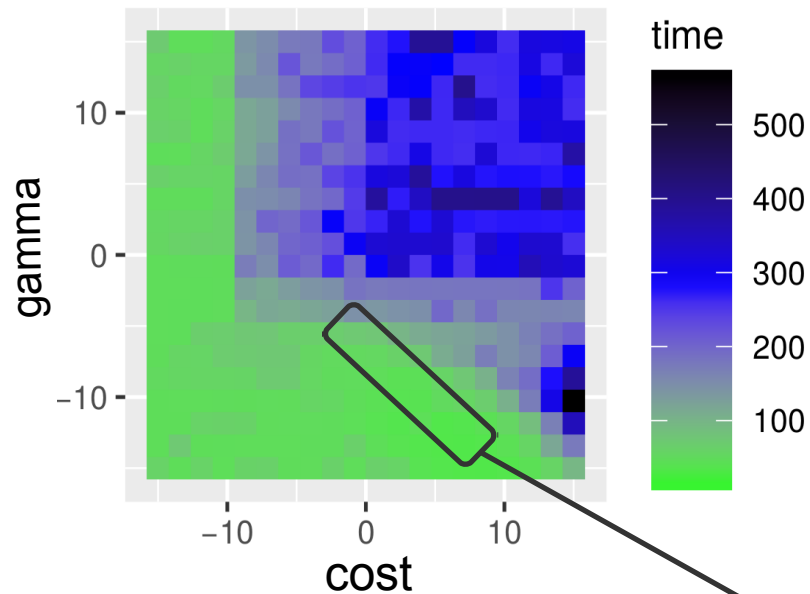




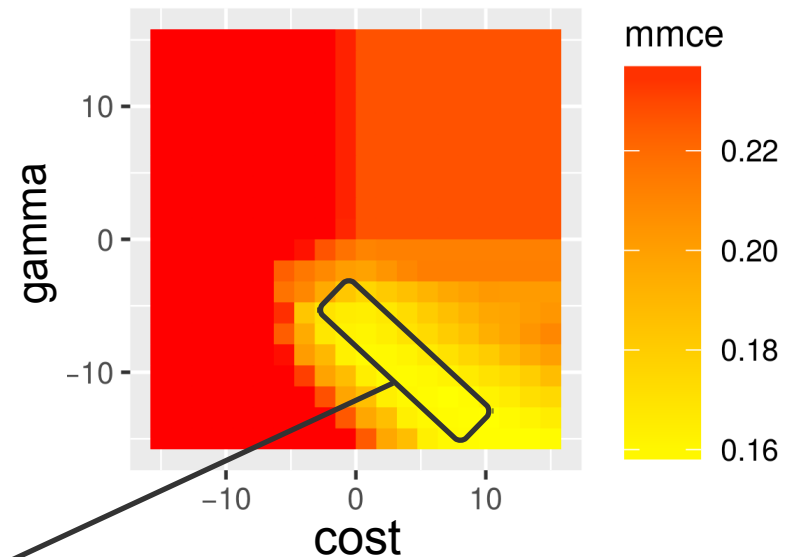
# Performance Estimation to Prioritize Parallel Jobs



## Runtime



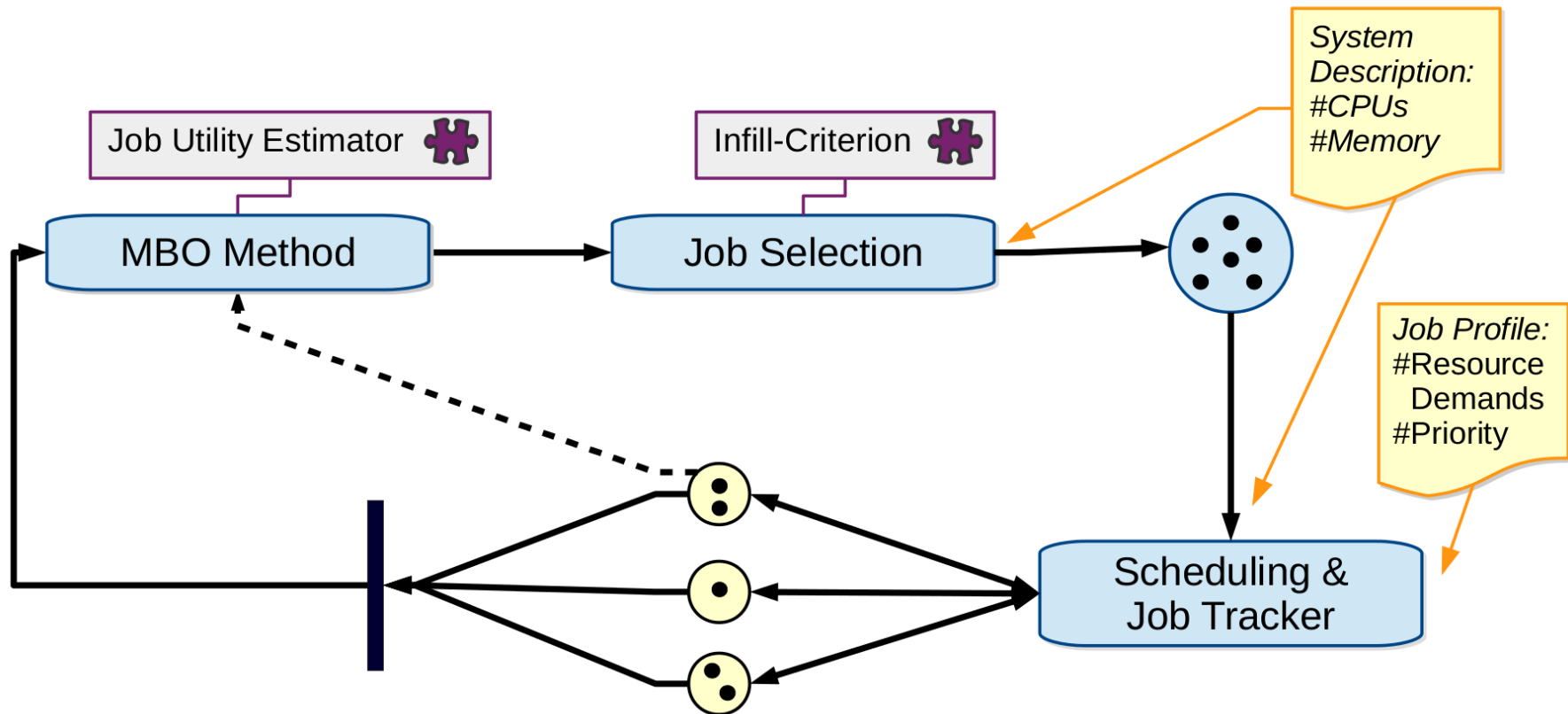
## Classification Error: Performance



Short Runtime  
High Performance



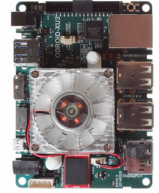
# Resource-Aware Model-Based Optimization



*H. Kotthaus et. al.: RAMBO: Resource-Aware Model-Based Optimization with Scheduling for Heterogeneous Runtimes and a Comparison with Asynchronous Model-Based Optimization. Learning and Intelligent Optimization 2017 (LION 11) (accepted for publication)*

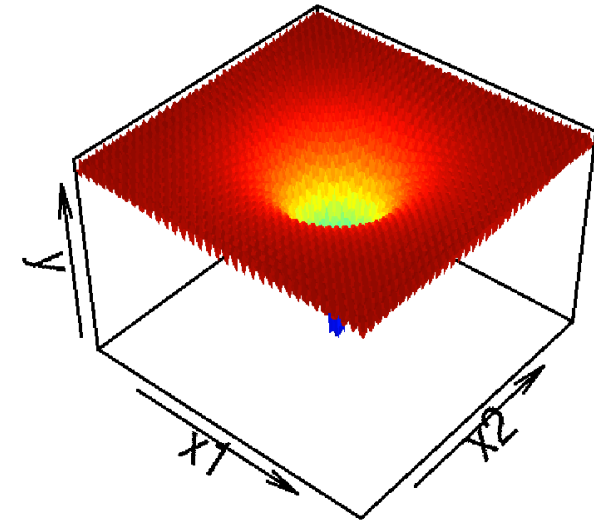


# Benchmark for the Heterogeneous Mobile Architecture Odroid



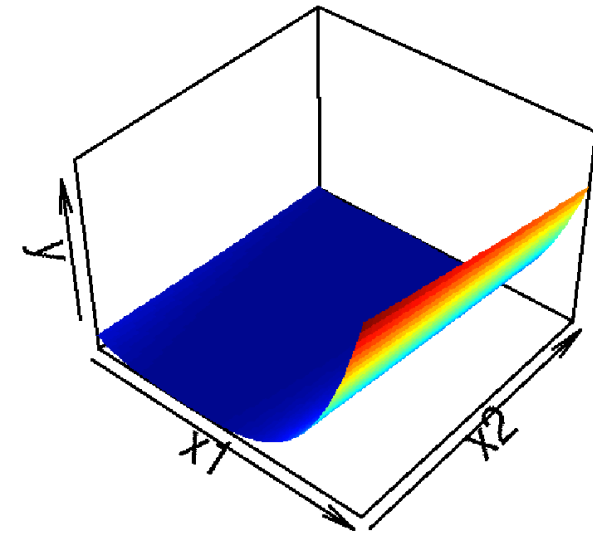
## *Objective Function*

- ▶ Ackley function
- ▶ Highly multi modal
- ▶ Goal: find the parameter configuration that produces the smallest  $y$



## *Runtime Function*

- ▶ Rosenbrock function
- ▶ Smooth surface simulates execution times of parallel jobs



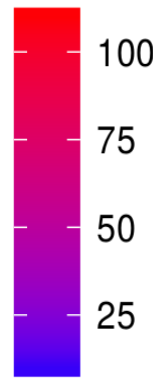
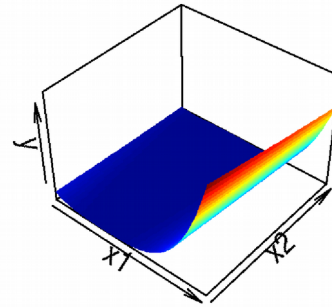
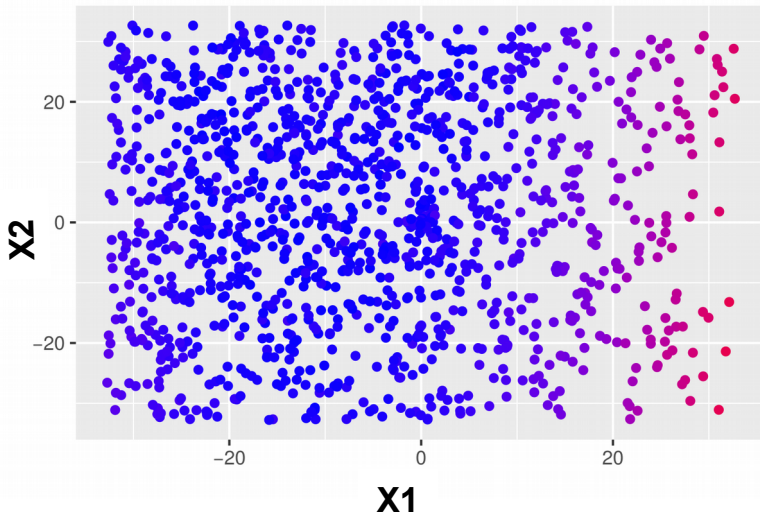
# Runtime Estimation via Regression Model

## Rosenbrock 2D Function on Odroid

**Fast CPU Cortex A15**

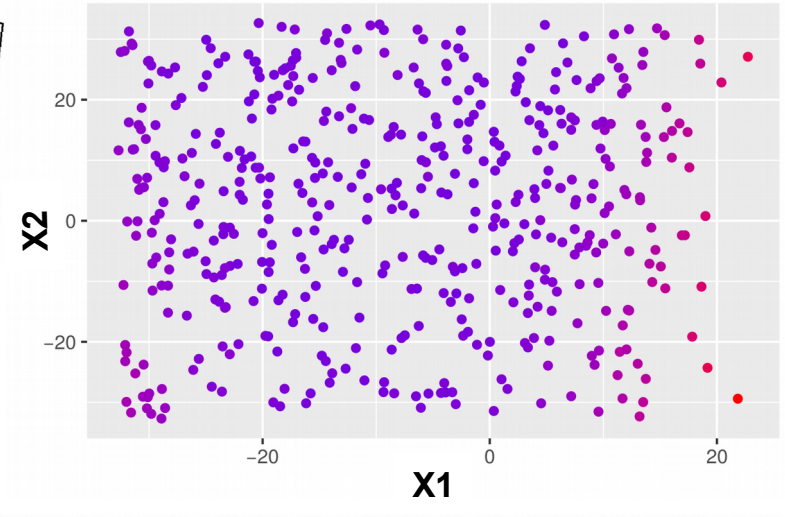
**Slow CPU Cortex A7**

Executed Runtime

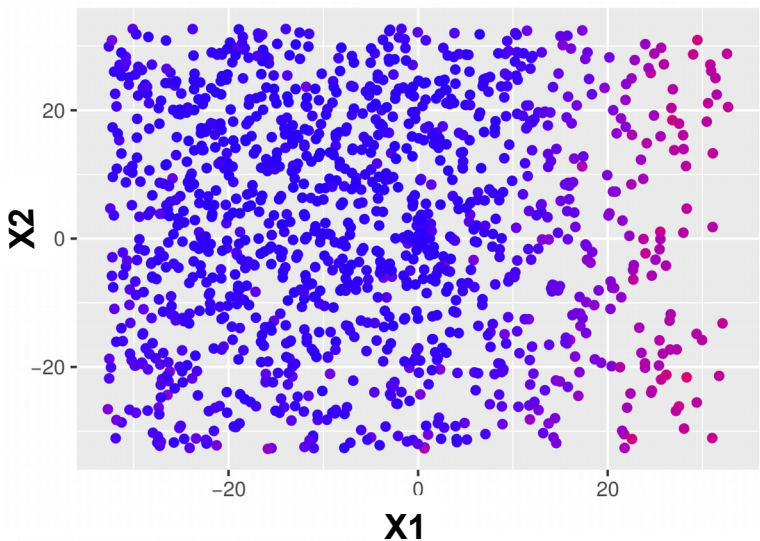


Runtime of  
evaluated  
configurations

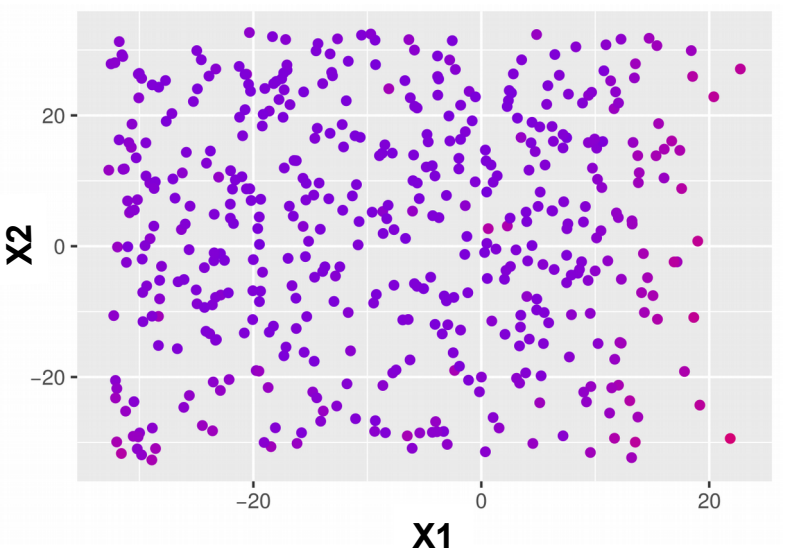
Executed Runtime



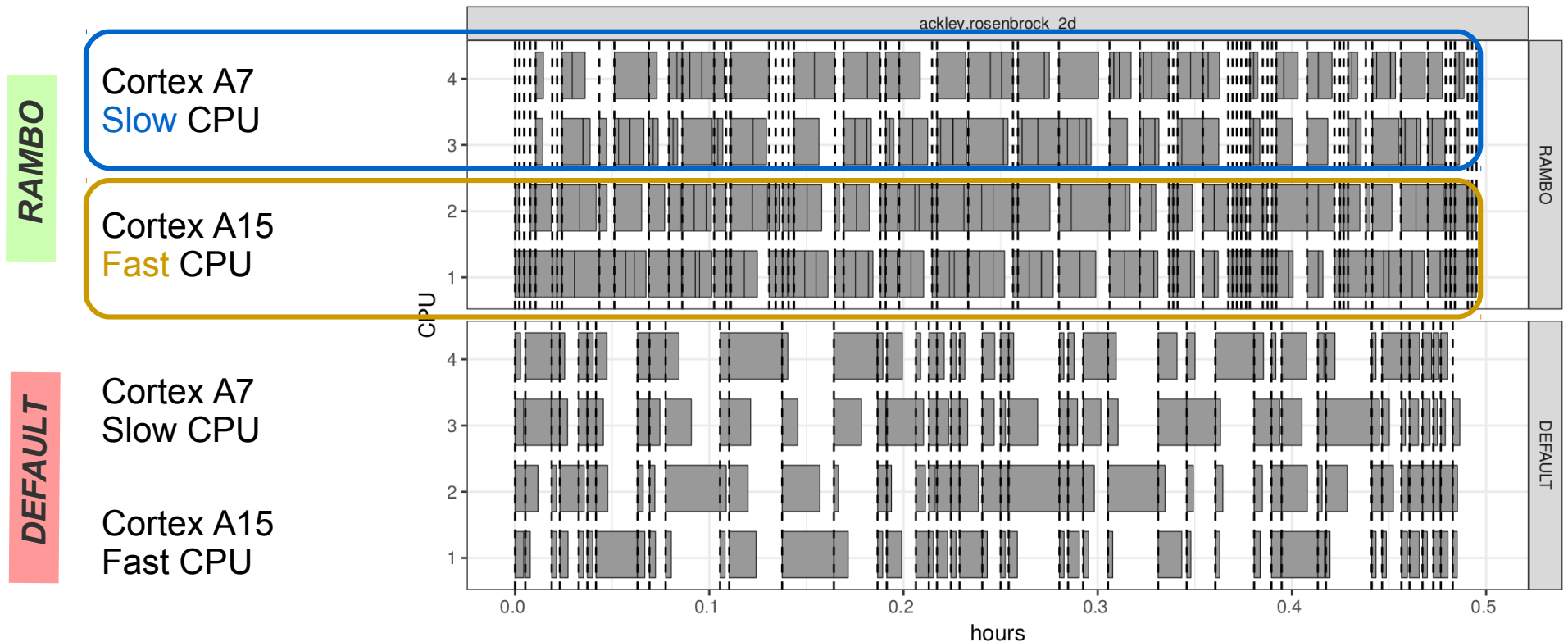
Estimated Runtime



Estimated Runtime

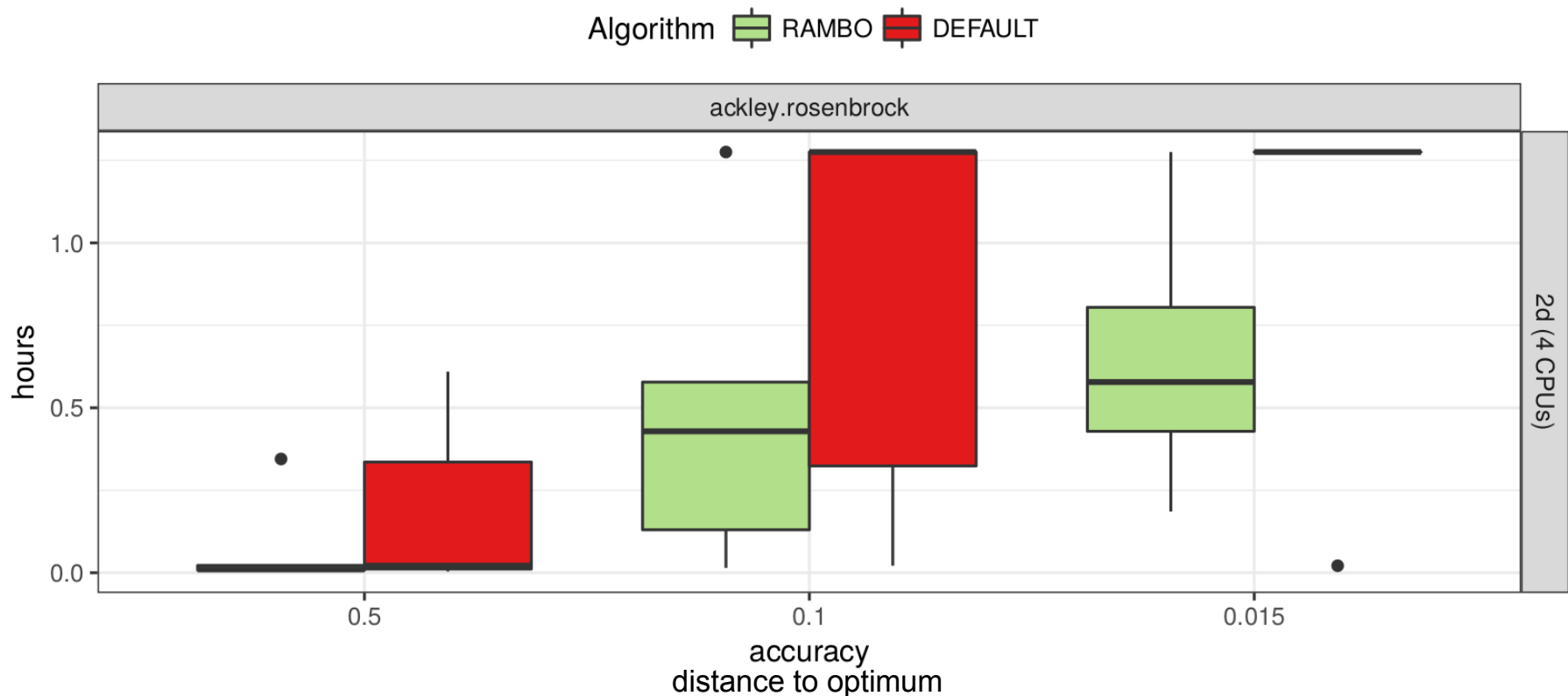


# Scheduling Snippet



→ ***RAMBO manages to balance parallel jobs more evenly on heterogeneous architectures***

# Who Finds the Best Configuration First?



→ ***RAMBO converges faster to the optimum (lower is better) on the heterogeneous architecture***

# Summary

---



## *Efficient Scheduling for Parallel R Programs on Heterogeneous Embedded Systems*

- ▶ CPU affinity parameter to allocate parallel jobs to specific CPUs
- ▶ Model for estimating execution times for different processor types
- ▶ Faster parallel machine learning on heterogeneous architectures

*We are also on github:*

- ▶ TraceR Profiling for Parallel R Programs  
→ <https://github.com/allr/tracer>
- ▶ Benchmarks  
→ <https://github.com/allr/benchR>
- ▶ RAMBO – Resource-Aware Model-Based Optimization  
→ [https://github.com/mlr-org/mlrMBO/tree/smart\\_scheduling](https://github.com/mlr-org/mlrMBO/tree/smart_scheduling)

