



# Adaptive Parallel Mergesort in Rust

Quentin Guilloteau

Monday 17th June, 2019

Supervisor: Frederic Wagner (LIG)

# Outline

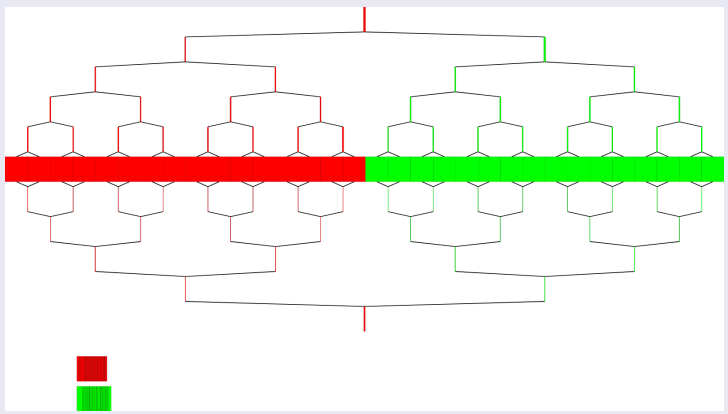
# Introduction

# Introduction

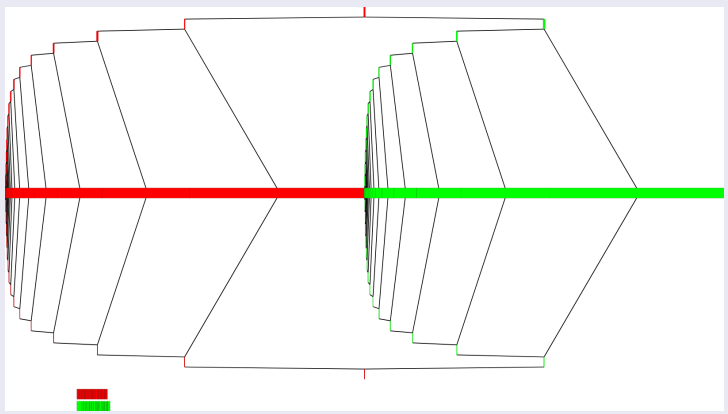
## Motivations

Fork-Join $\rightarrow$ Work-Stealing/Task-based $\rightarrow$ Rayon-Adaptive
This internship: impact of the abstraction on performance ?

## Schedulers



(a) `Join`



(b) `Join-Context`

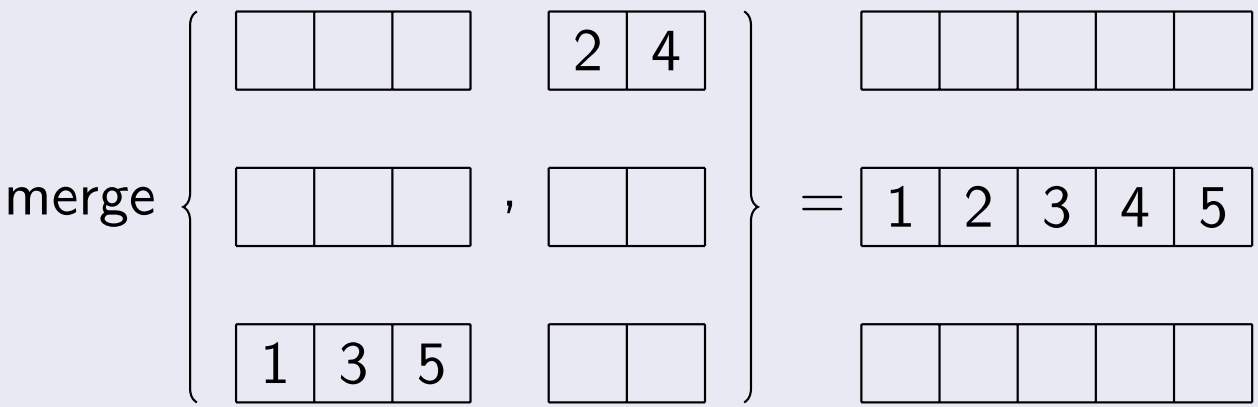
# Presentation of the Algorithm

# Presentation of the Algorithm

## Algorithm

Classical "divide and conquer" hybrid Mergesort using TimSort

## Data Structure



# Optimizations

# Optimizations

## Outline

Before comparing parallel sorts: let's optimize !

1. Optimize time
2. Optimize memory usage

# Time Optimization

# Optimizations - Extra `memcpy`

## Problem

First buffer points to user array.
Output data not necessary in the first buffer at the end.

## Idea

Try to use only two buffers, rely on the third one if needed.

## Rule when Merging

- even depth $\rightarrow$ merge in buffer 2, if not possible buffer 3
- odd depth $\rightarrow$ merge in buffer 1, if not possible buffer 3

# Optimizations - Extra `memcpy`

## Idea

Try to use only two buffers and rely only on the third one if needed.

## Finding the new block size

Let $\mathcal{A}(B)$ be a $k$-way mergesort. Let $n$ be the size of the input. Height of the merging tree:

$$H = \left\lceil \log_k \left( \frac{n}{B} \right) \right\rceil \rightarrow H' = \left\lceil \frac{H}{2} \right\rceil \times 2 \rightarrow B' = \left\lceil \frac{n}{k^{H'}} \right\rceil \quad (1)$$

# Memory Usage Optimization

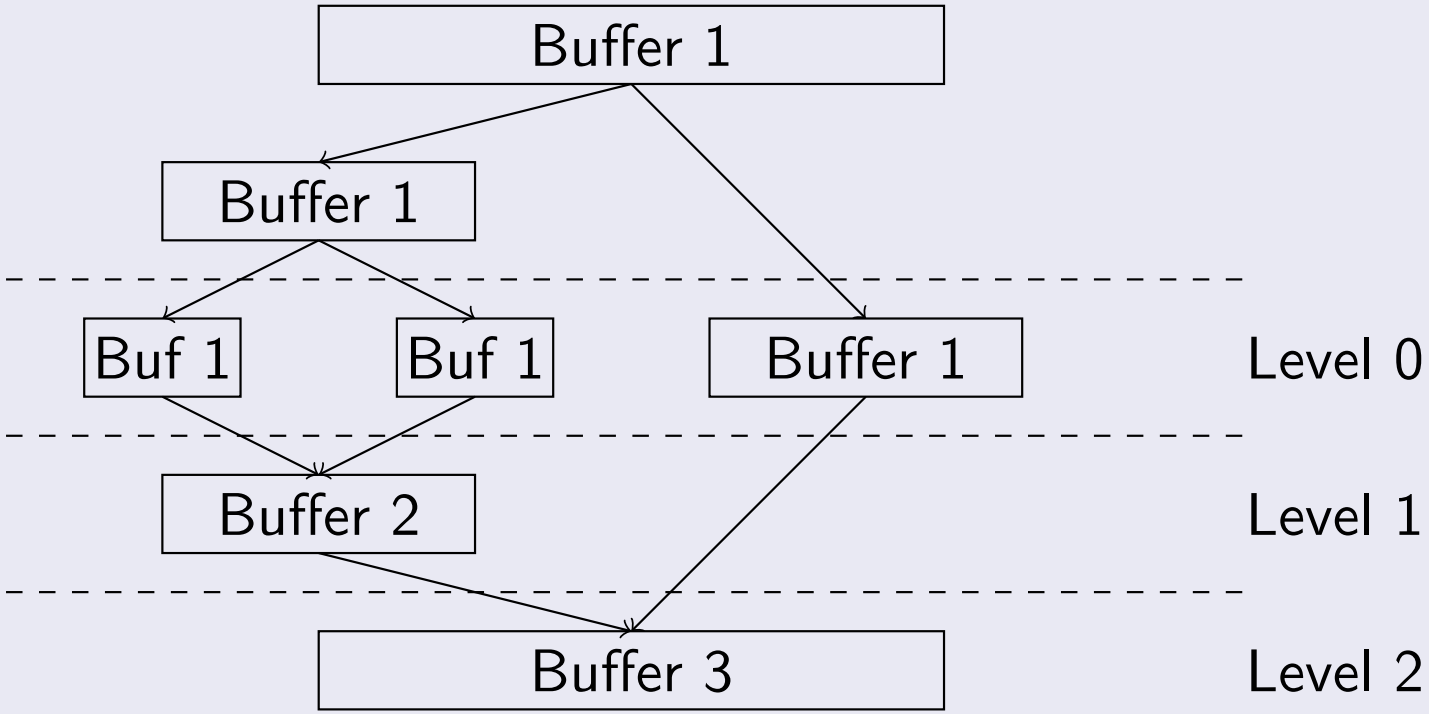
# Optimizations - 2 Buffers

## Problem

Non negligible memory cost: $3 \times 400$Mb for 100M `uint32_t`

## Situations where 3 buffers are needed

3rd buffer required when merging data from different depths parity.



# Optimizations - 2 Buffers

## Solution: `Join`

If array of size $n$ and $k$-way mergesort:

$$n = (k-1) \times n_1 + n_2 \implies |n_1 - n_2| \leq k - 1 \tag{2}$$

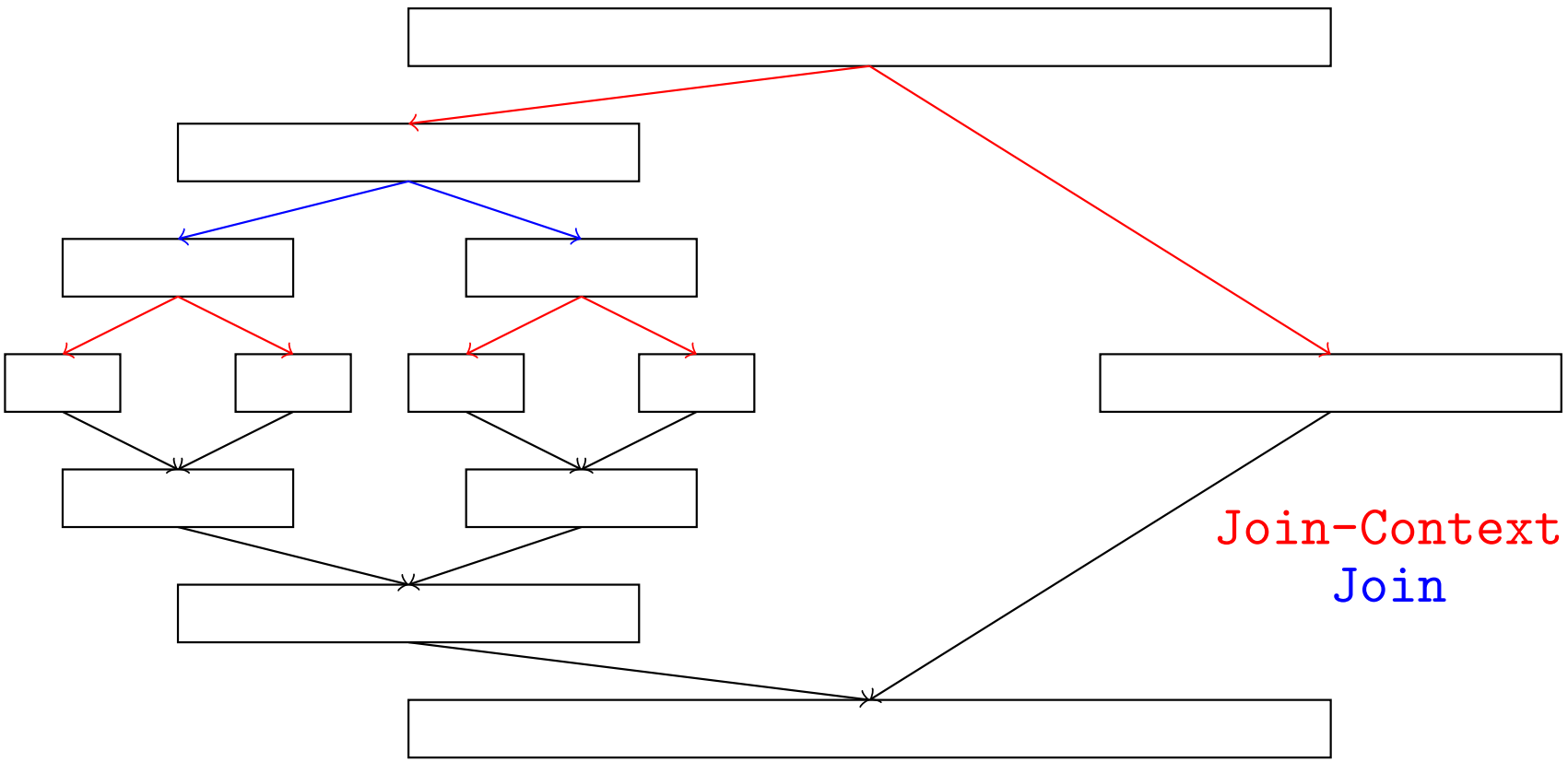
Limiting case: $n_2 > B \geq n_1 \implies B + (k-1) \geq n_2 > n_1$
Increase the initial block size by $k - 1$

## Solution: `Join-Context`

Alternate the calls to the schedulers:

- Even $\rightarrow$ use `Join-Context`
- Odd $\rightarrow$ use `Join`

# Optimizations - 2 Buffers: `Join-Context`



# Performances

# Performances

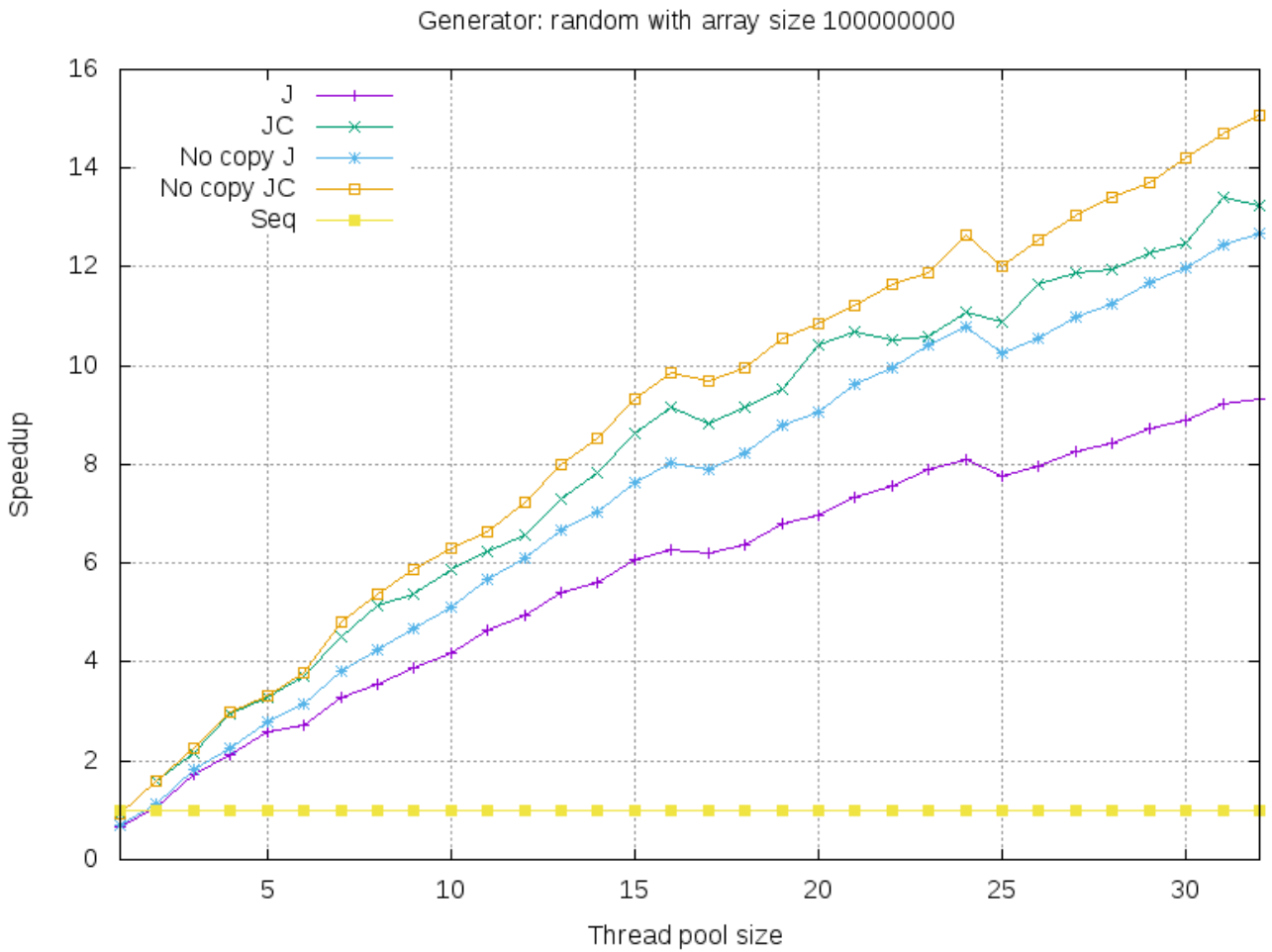



Figure: Comparison between schedulers

# Performances

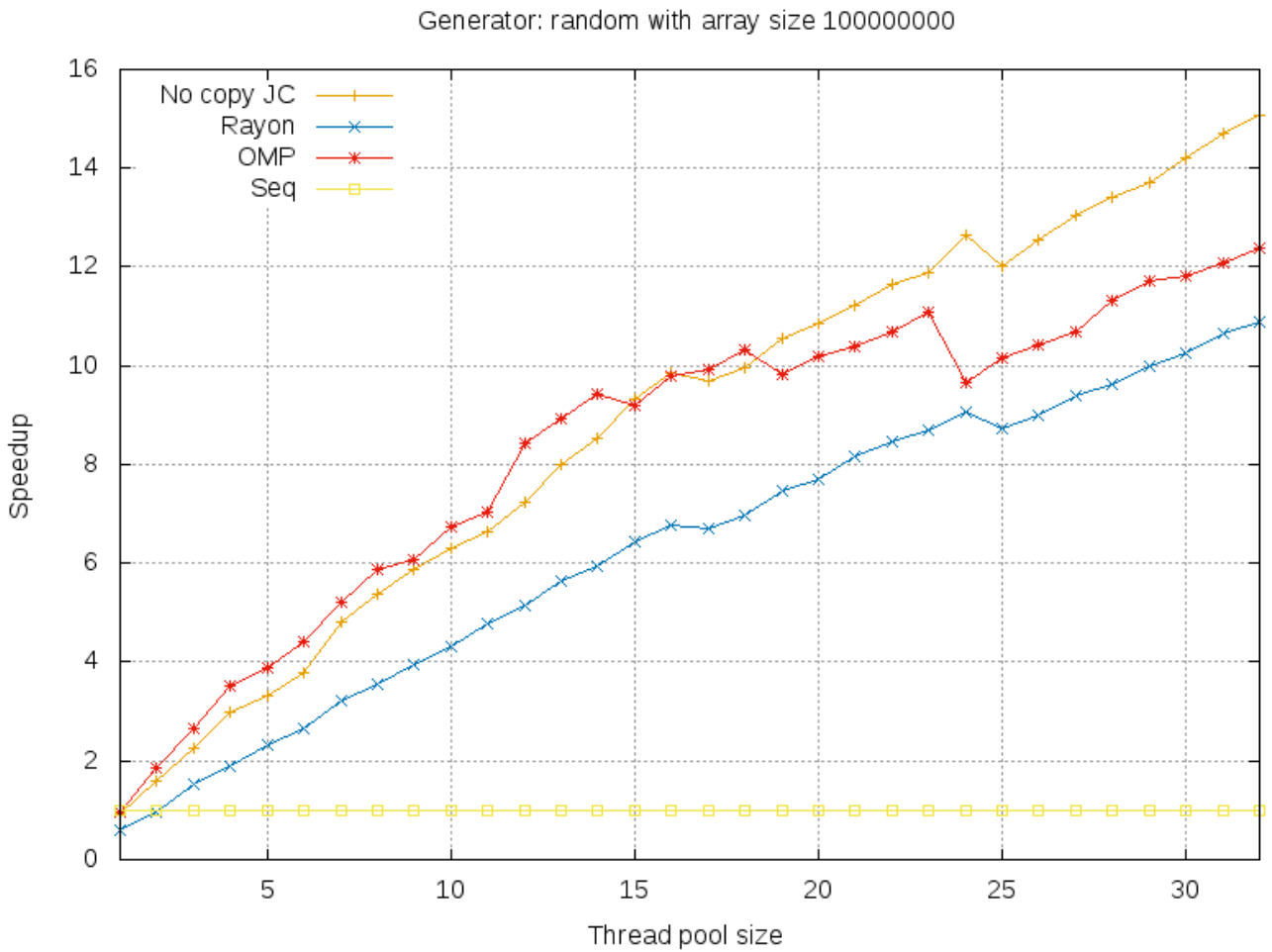



Figure: Comparison with Rayon and OMP

# 3-way Mergesort

# 3-way Mergesort

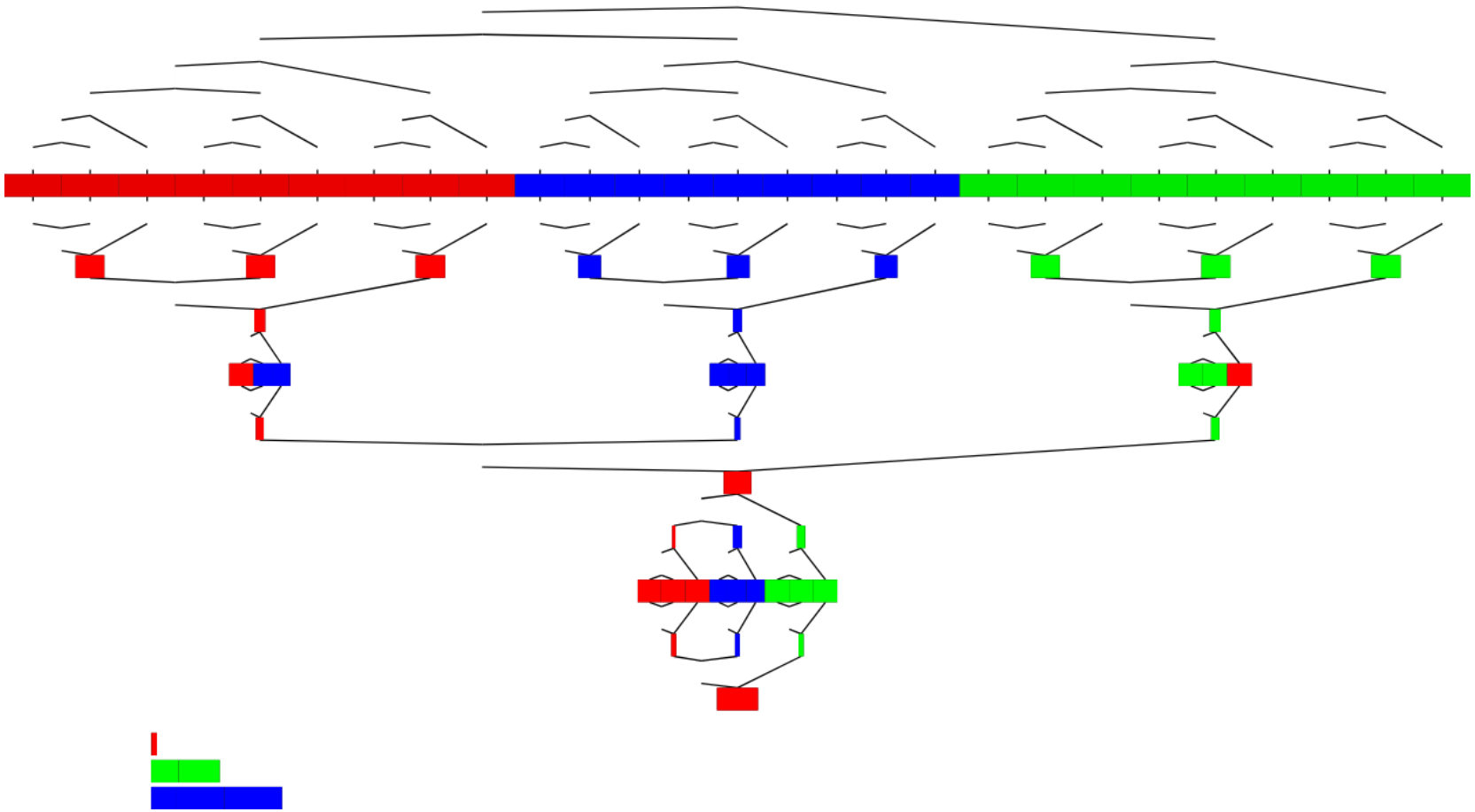


Figure: Split in 3 with 3 threads

# 3-way Mergesort: Performances

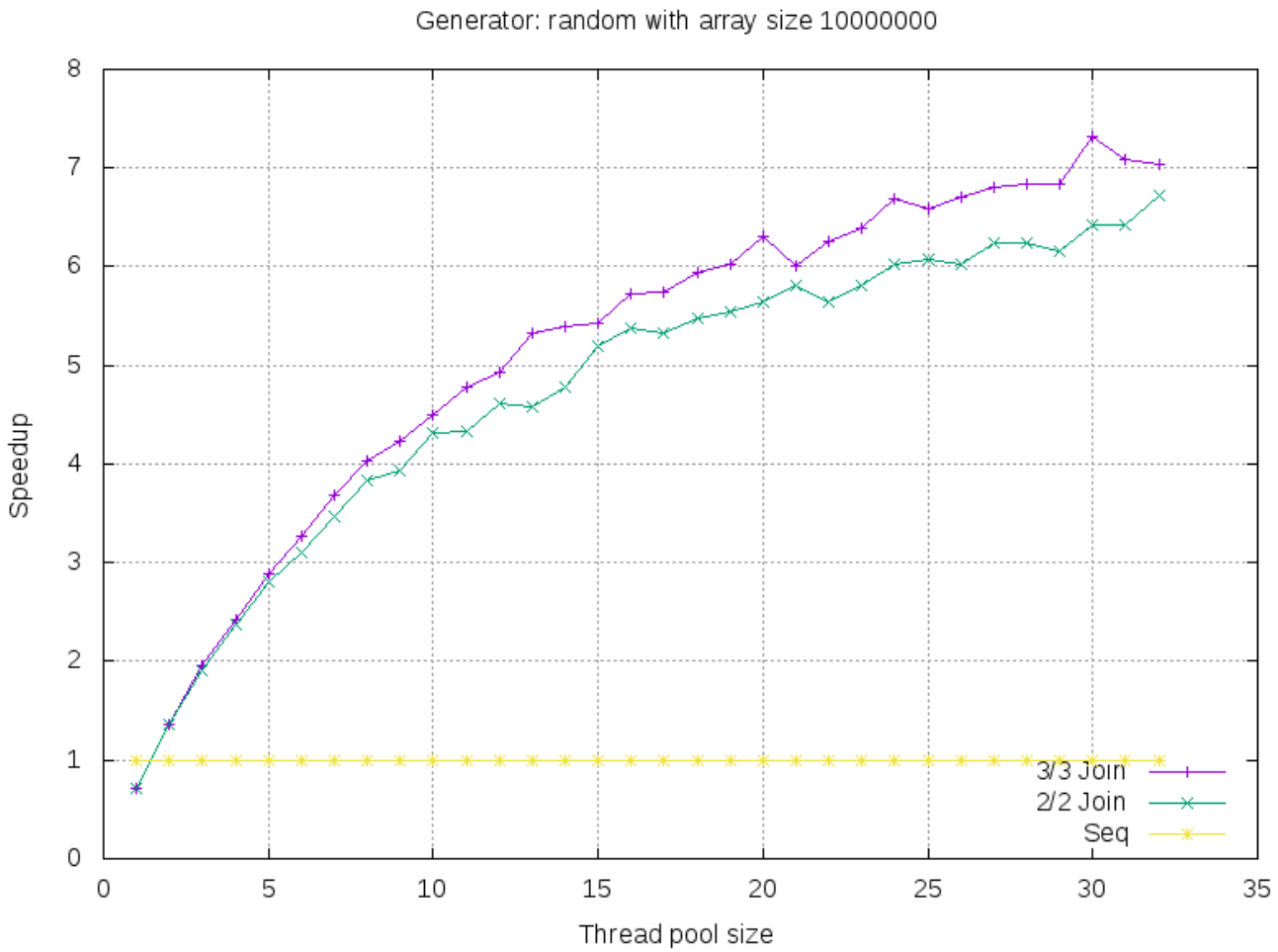



Figure: Comparison between schedulers

# Conclusion & Future Work

# Conclusion & Future Work

## Conclusion

- Manage to beat/compete with standard parallel sorts
- Room for improvement: reversed arrays
- Still a preliminary work: $k$-way mergesort

# CTRL-A - CTRL-CiGri

# CTRL-A - CTRL-CiGri

## Title

Minimizing Cluster Under-Use with a Control-Based Approach

## Some notions

- OAR: Scheduler of the computing clusters
- CiGri: Lightweight grid system which exploits the unused resources of a set of computing clusters

## The idea

Feed the information from OAR into a feedback loop, to control how CiGri behaves in order to maximize the utilization of the resources, and to avoid overload.

# Questions ?

Thank you for your attention.
Time for Questions !

# 3-way Mergesort: Performances

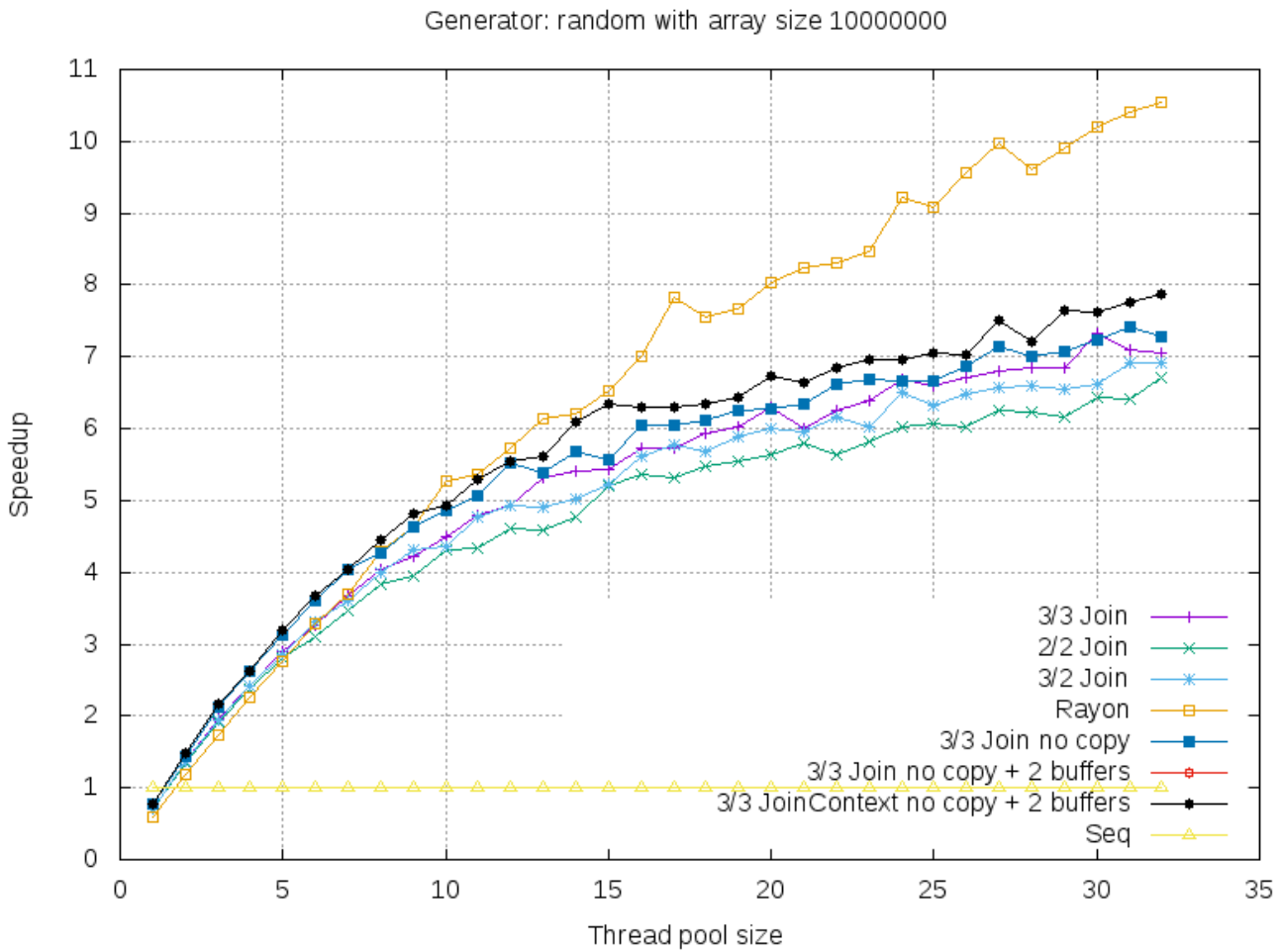



Figure: Comparison between schedulers

# 2-way Mergesort: Sorted arrays

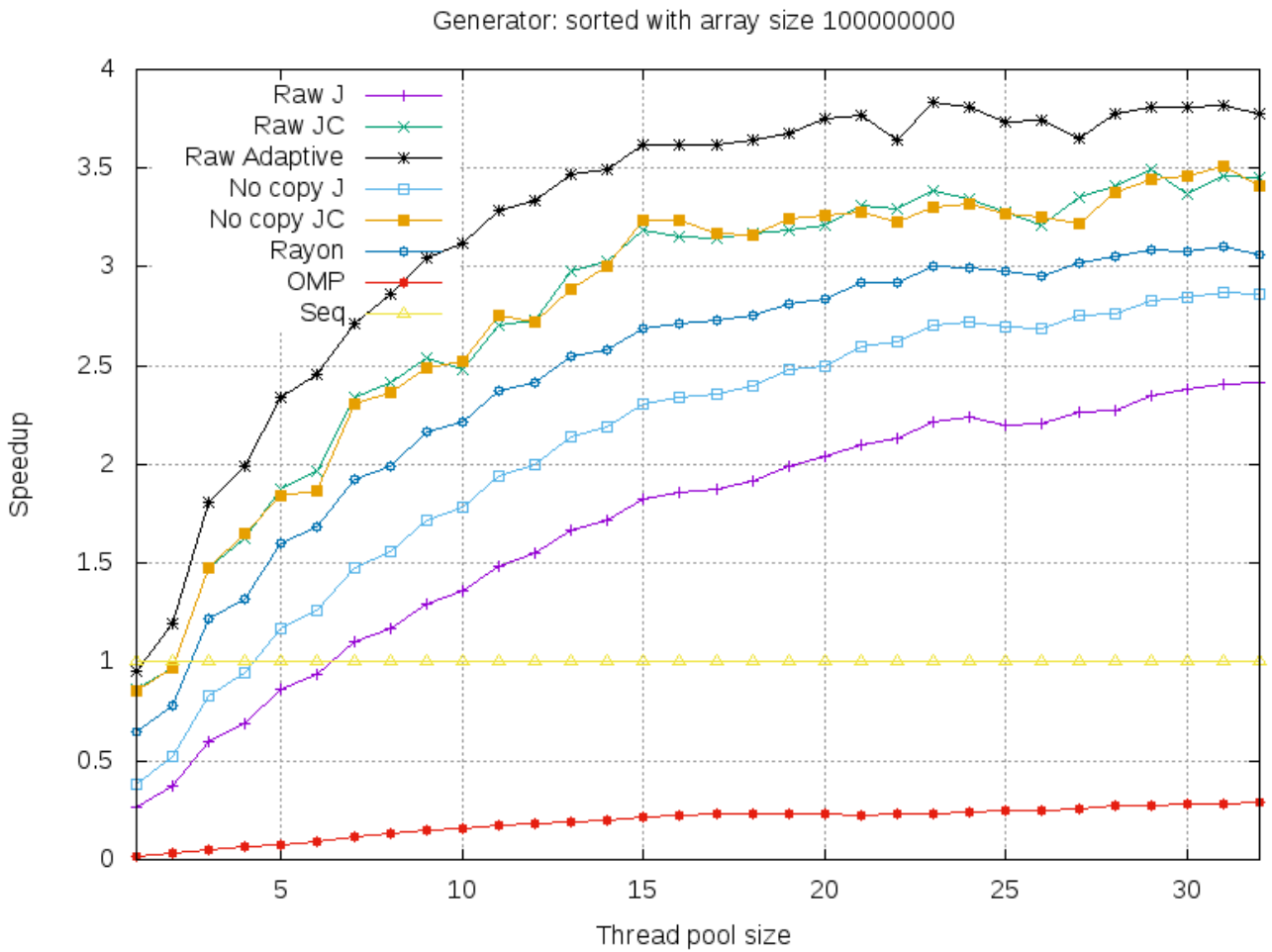



Figure: Comparison between schedulers

# 2-way Mergesort: Reversed arrays

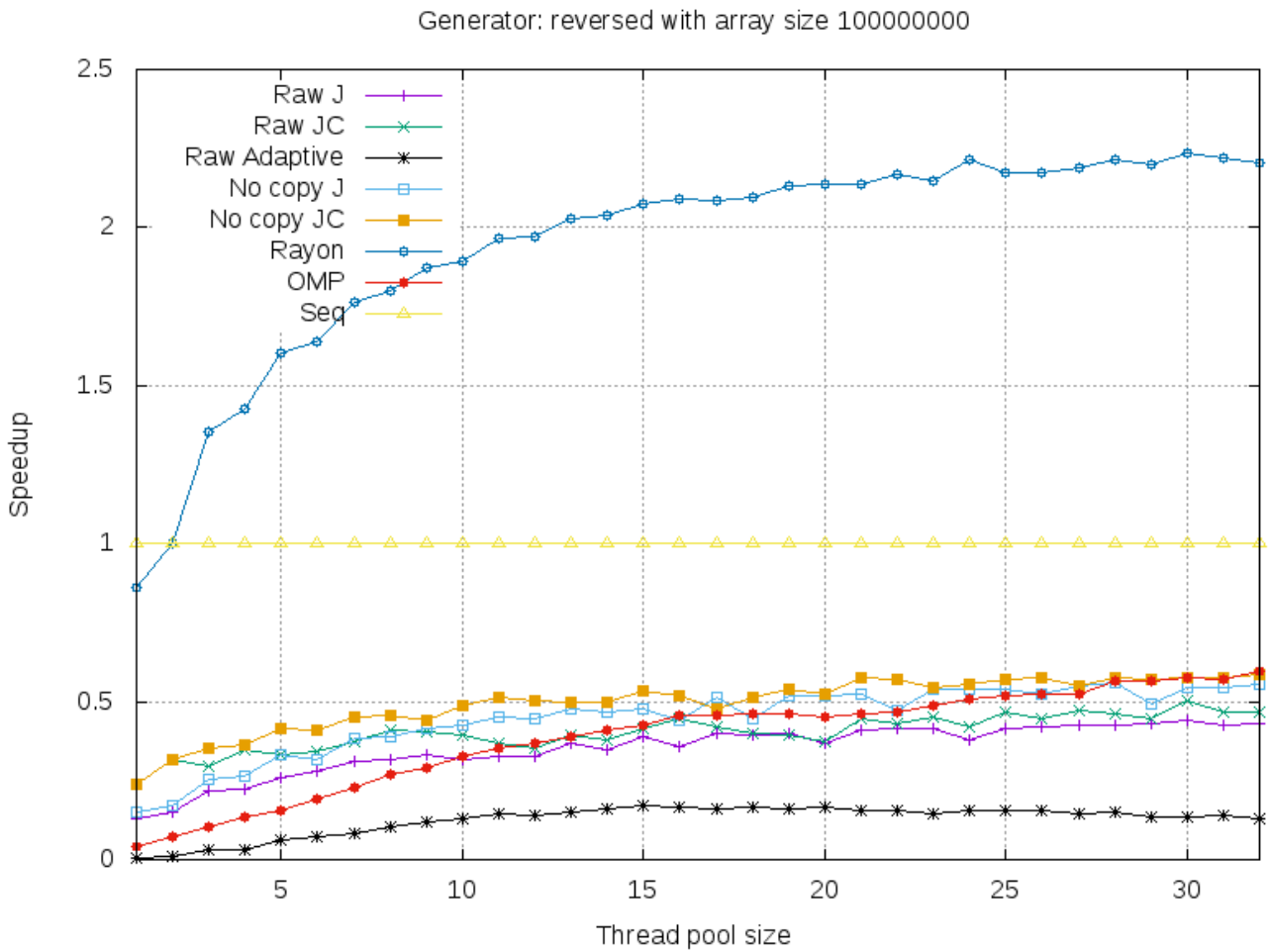



Figure: Comparison between schedulers

# 2-way Mergesort: Random arrays

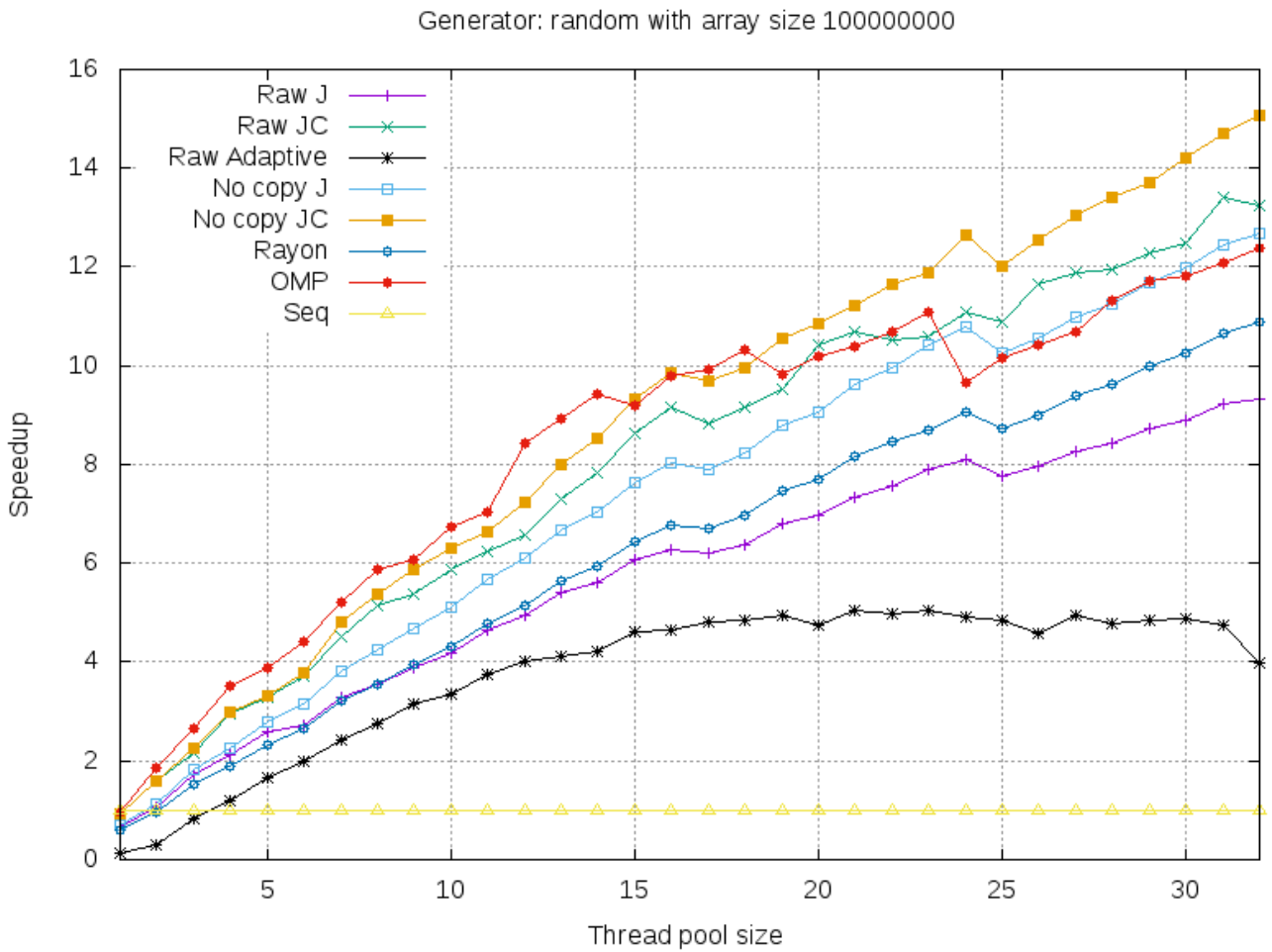



Figure: Comparison between schedulers