A parallel framework for Genetic Algorithms in Haskell

Matthias Delbar

Overview

1. Introduction
2. Parallelism
3. Framework
4. Parallelization
5. Evaluation
6. Future work

Overview - Introduction

1. Introduction
2. Parallelism
3. Framework
4. Parallelization
5. Evaluation
6. Future work

Overview - Introduction

1. Introduction

- Genetic algorithms
- Why Haskell?

Genetic algorithms

- Search heuristic based on theory of evolution
  - Crossover, mutation, selection, fitness
- Small changes over many generations
  - Iteratively improve the solution
- Not always the optimal solution
  - *Acceptable* solution in *acceptable* time

Why Haskell?

- ... it is a functional programming language
  - the same input gives the same output
- ... it supports higher order functions
  - “pass functions as parameter”
- ... it supports parallelism
  - different approaches

Overview - Parallelism

1. Introduction
2. Parallelism
3. Framework
4. Parallelization
5. Evaluation
6. Future work

Overview - Parallelism

2. Parallelism

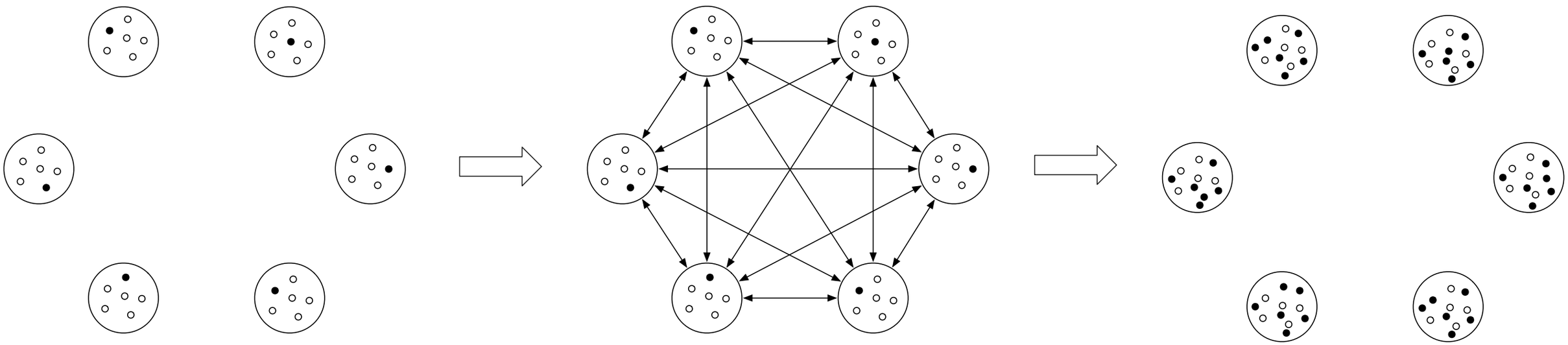
- GAs and parallelism
- Haskell and parallelism

GAs and parallelism

- Selection in parallel
  - Depends on selection strategy
- Mutation and crossover in parallel
  - Depends on chosen parameters and strategies
- Distributed method
  - Independent from parameters or strategies

Distributed method

- Multiple islands, each with its own population
- Every island executes in parallel
- After X generations, cross-pollination takes place



Haskell and parallelism

- Haskell functions `` `par` `` and `` `pseq` ``
  - Runtime decides if thread is created
- Implicit data parallelism
  - Data Parallel Haskell
- Explicit data parallelism
  - Manually split population, then use `` `par` `` en `` `pseq` ``

Overview - Framework

1. Introduction
2. Parallelism
3. Framework
4. Parallelization
5. Evaluation
6. Future work

Overview - Framework

3. Framework
    - Original framework
    - Initial changes
    - Refactoring

Original framework

- Multiple published frameworks
  - Problem: no code available or missing features
- GA.hs by Kenneth Hoste
  - Generic, code available, good starting point
  - Made some initial changes, and further refactoring

Initial changes

Monadic → Pure

Lazy → Strict

Memory issues → trimArchives

Checkpointing → Not in scope

Refactoring

- Altered the structure of the framework
  - Better overview
- Perform selection separately
  - No longer as part of mutation/crossover
- Simpler (faster) selection strategy
  - Ideally this would be a parameter of the framework

Overview - Parallelization

1. Introduction
2. Parallelism
3. Framework
4. Parallelization
5. Evaluation
6. Future work

Overview - Parallelization

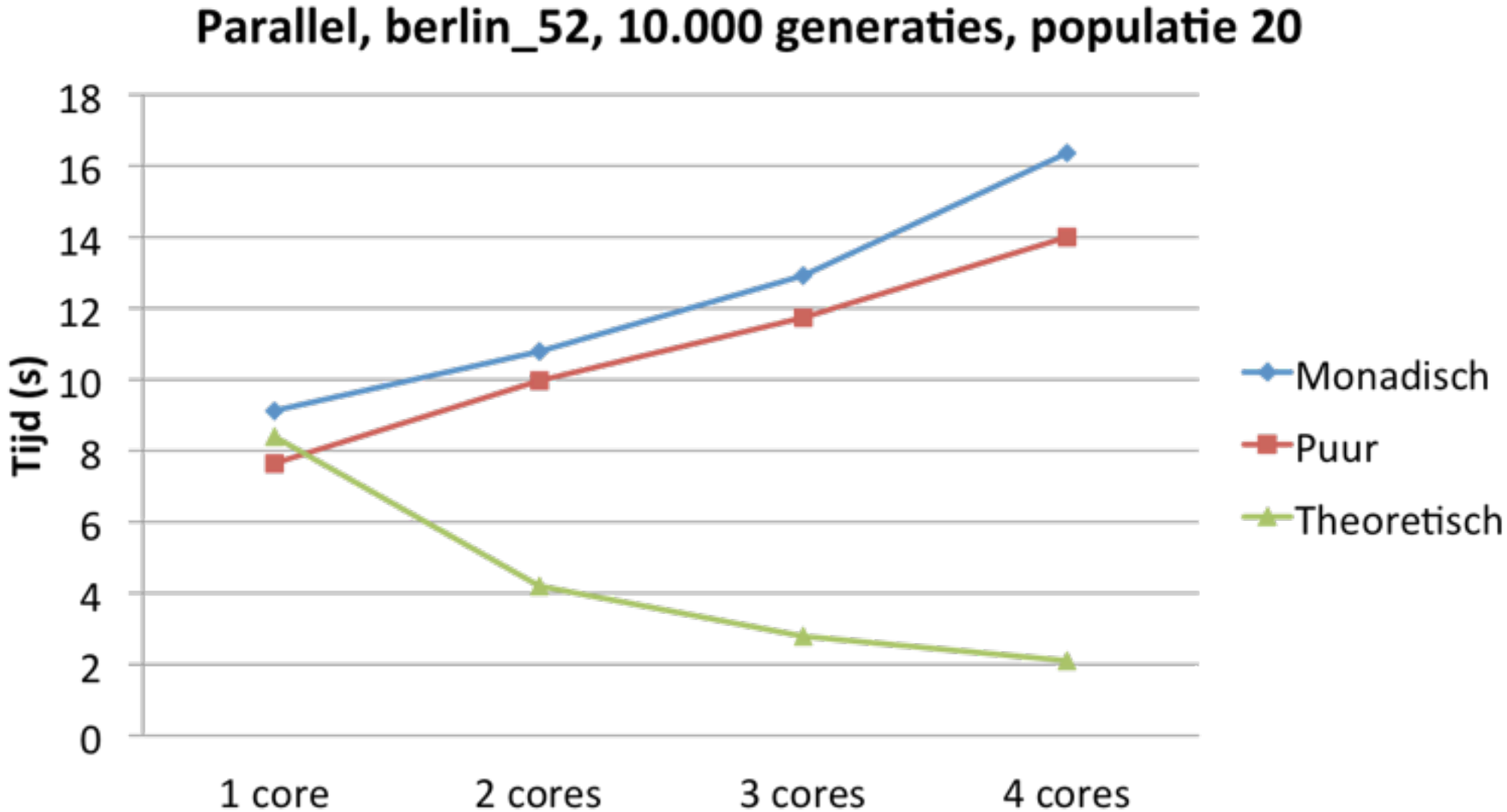
4. Parallelization

- Naive parallelism via `` `par` `` & `` `pseq` ``
- Data parallelism
- Distributed method

Naive parallelism

Crossover and mutation in parallel

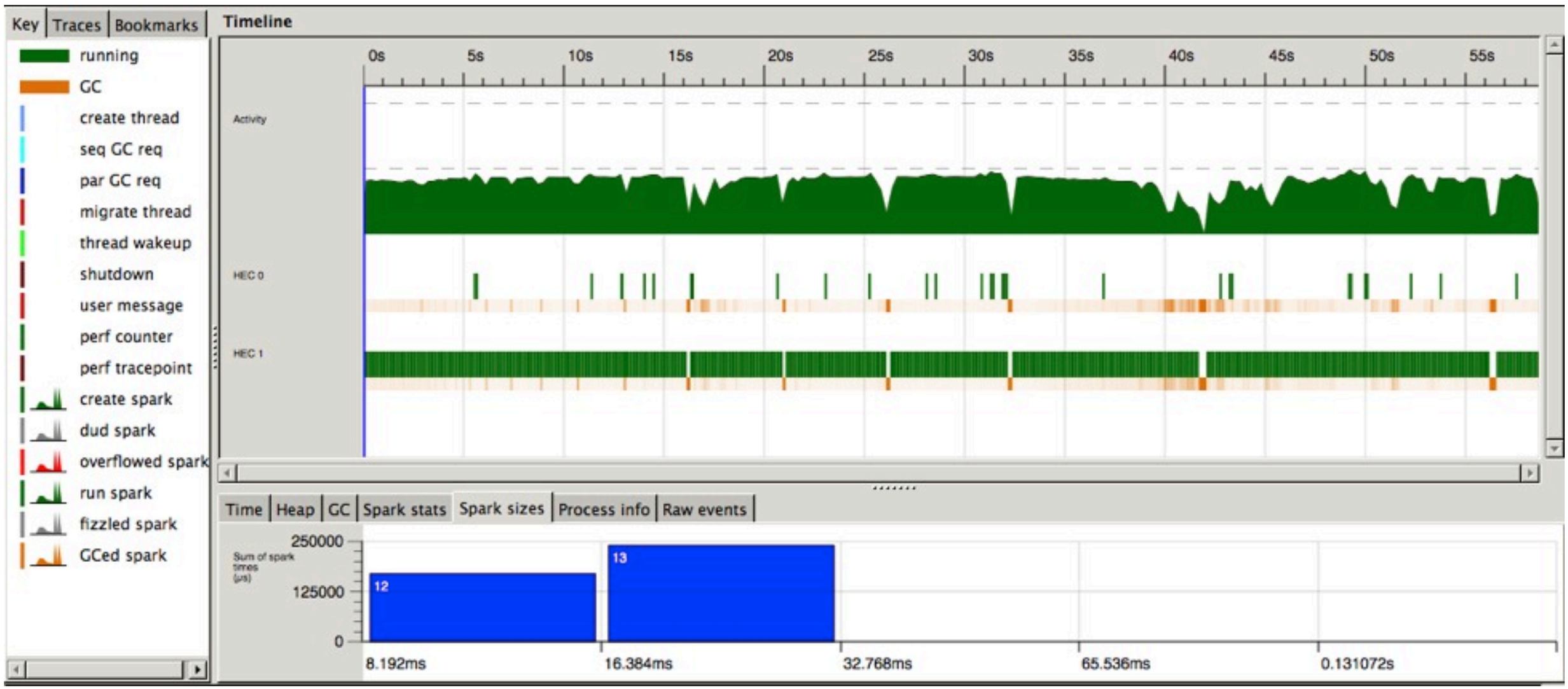
Barely any speedup, outweighed by overhead



Naive parallelism

Crossover and mutation in parallel

Too fine granularity (threads take too little time)



Data parallelism

- Data Parallel Haskell
  - Not mature or generic enough, yet
- Explicit data parallelism
  - Issues with granularity
  - Might work with extremely large populations

Distributed method

- Uses `par` en `pseq`
  - Every island executes in a separate thread
- Cross-pollination as a step in the framework
  - Happens every X generations, followed by selection
  - Multiple strategies possible
  - Copy the best entity from each island to all others

Overview - Evaluation

1. Introduction
2. Parallelism
3. Framework
4. Parallelization
5. Evaluation
6. Future work

Overview - Evaluation

5. Evaluation
    - Benchmarks
    - Fitness
    - Execution time

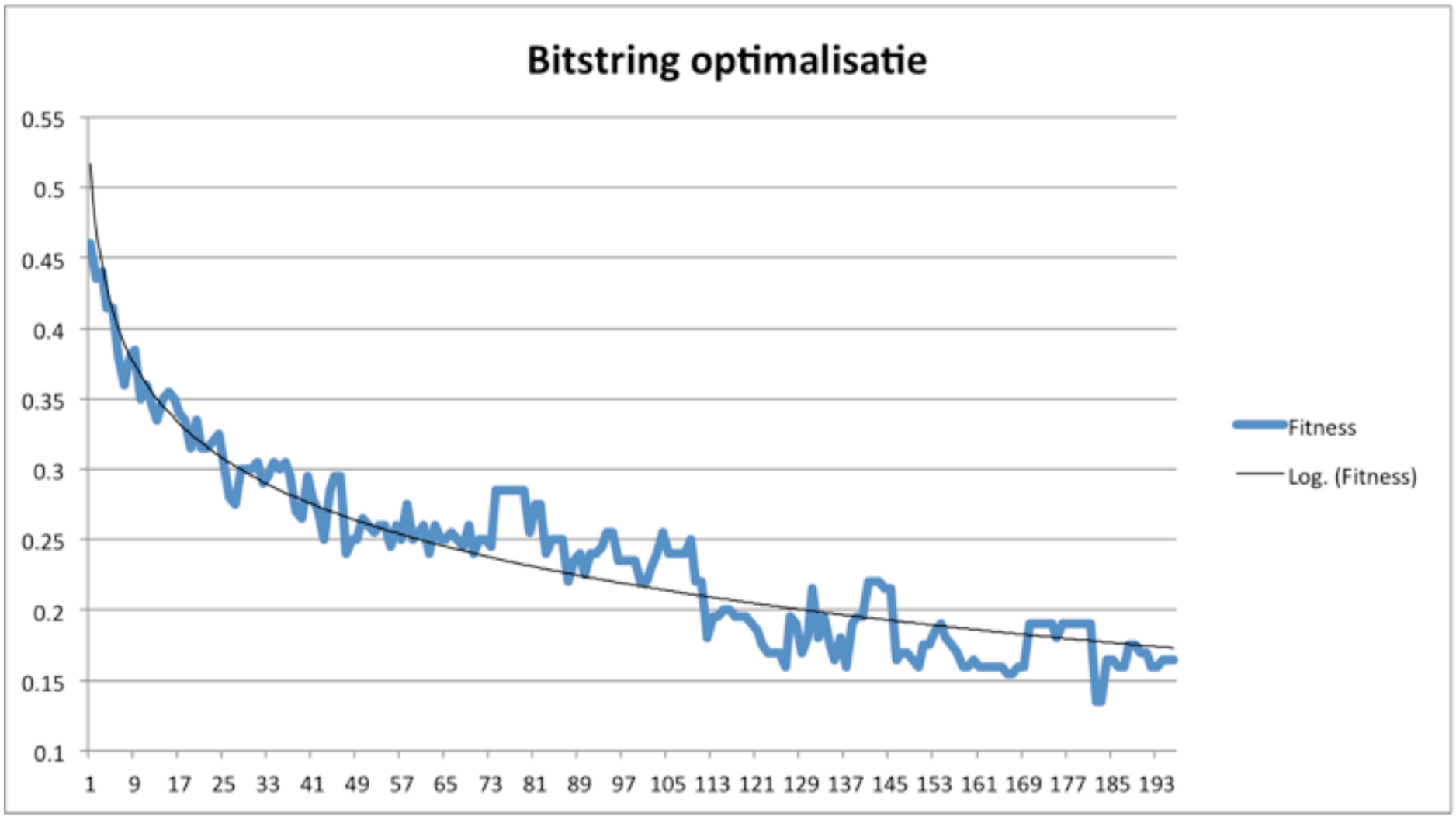
Benchmarks

- Travelling Salesman Problem
  - Famous (difficult) problem with well-known inputs
  - Results can depend on input
- Bit string optimization
  - Straightforward problem, not input-dependent
  - Used mostly for time measurements

Fitness

Rule of thumb: more islands → better fitness

Logarithmic scale

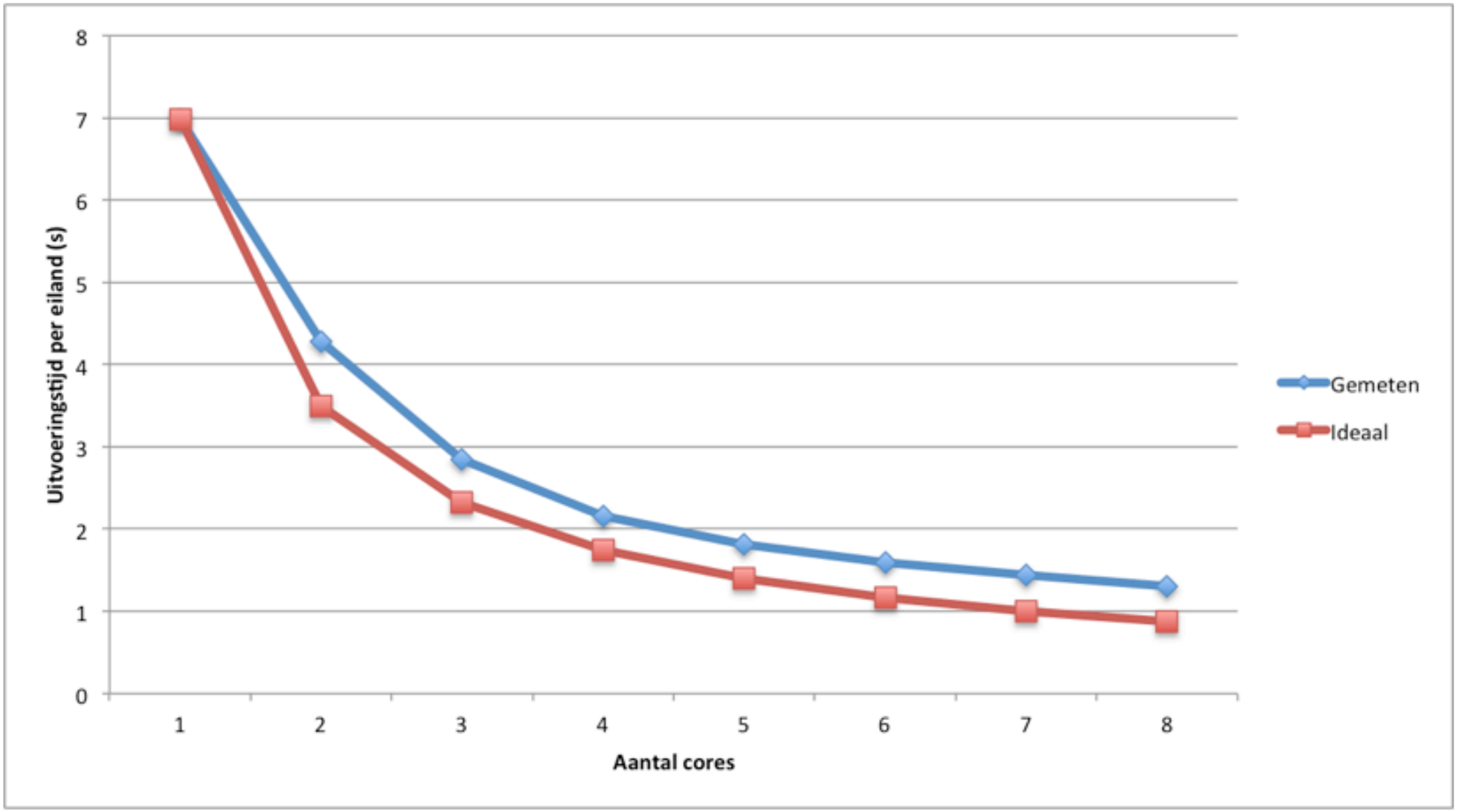


Execution time

- Speed gain of parallel implementation
- Overhead
  - Synchronization and cross-pollination overhead
- Framework performance
  - Impact of the number of islands or generations

Execution time

- Speed gain of parallel implementation



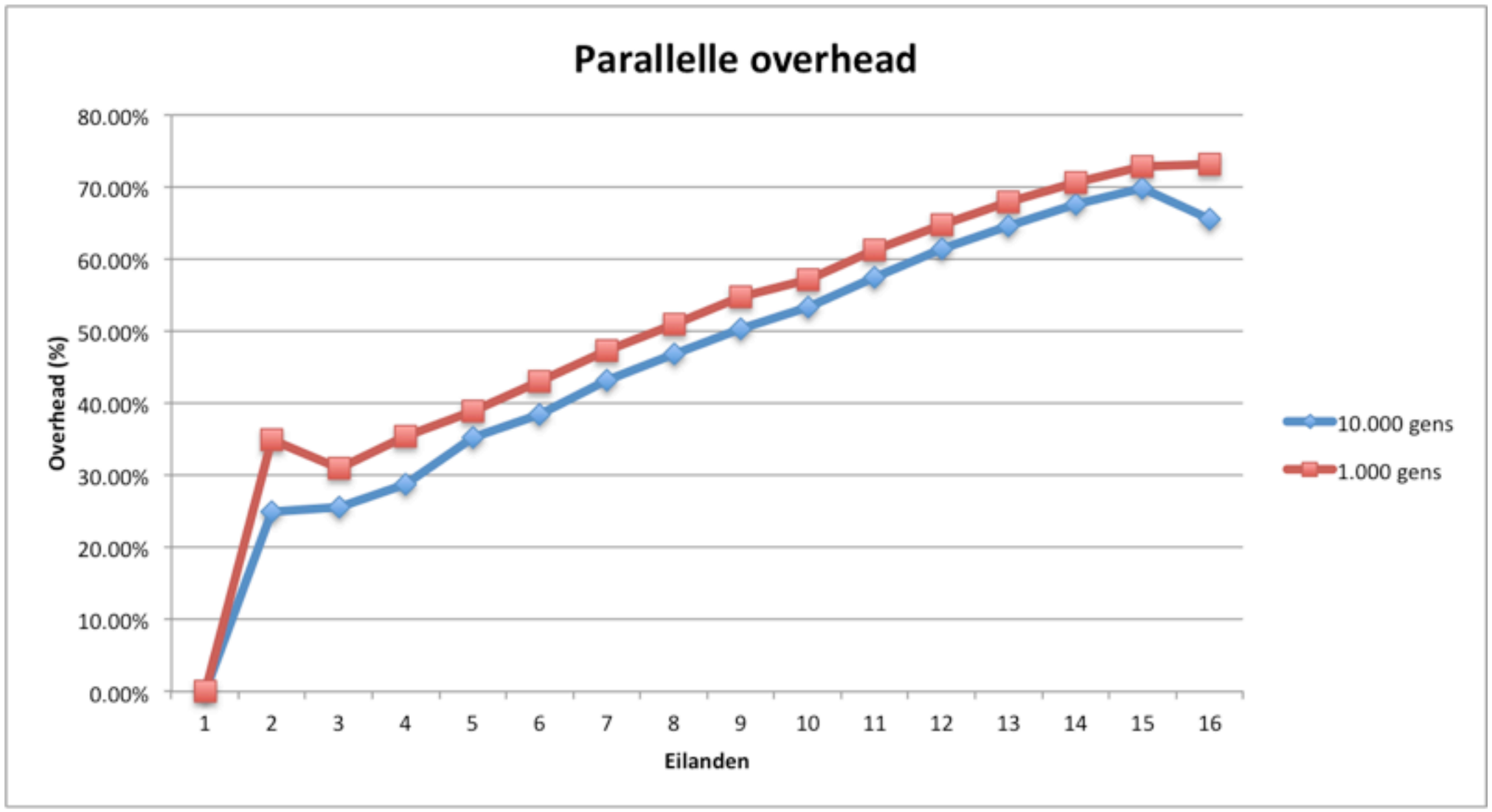
Overhead

- Synchronization overhead
  - Thread waits until other threads are finished
- Cross-pollination overhead
  - Added cost from the copying of entities
  - Not significant (even smaller than startup cost)

Overhead

Synchronization overhead

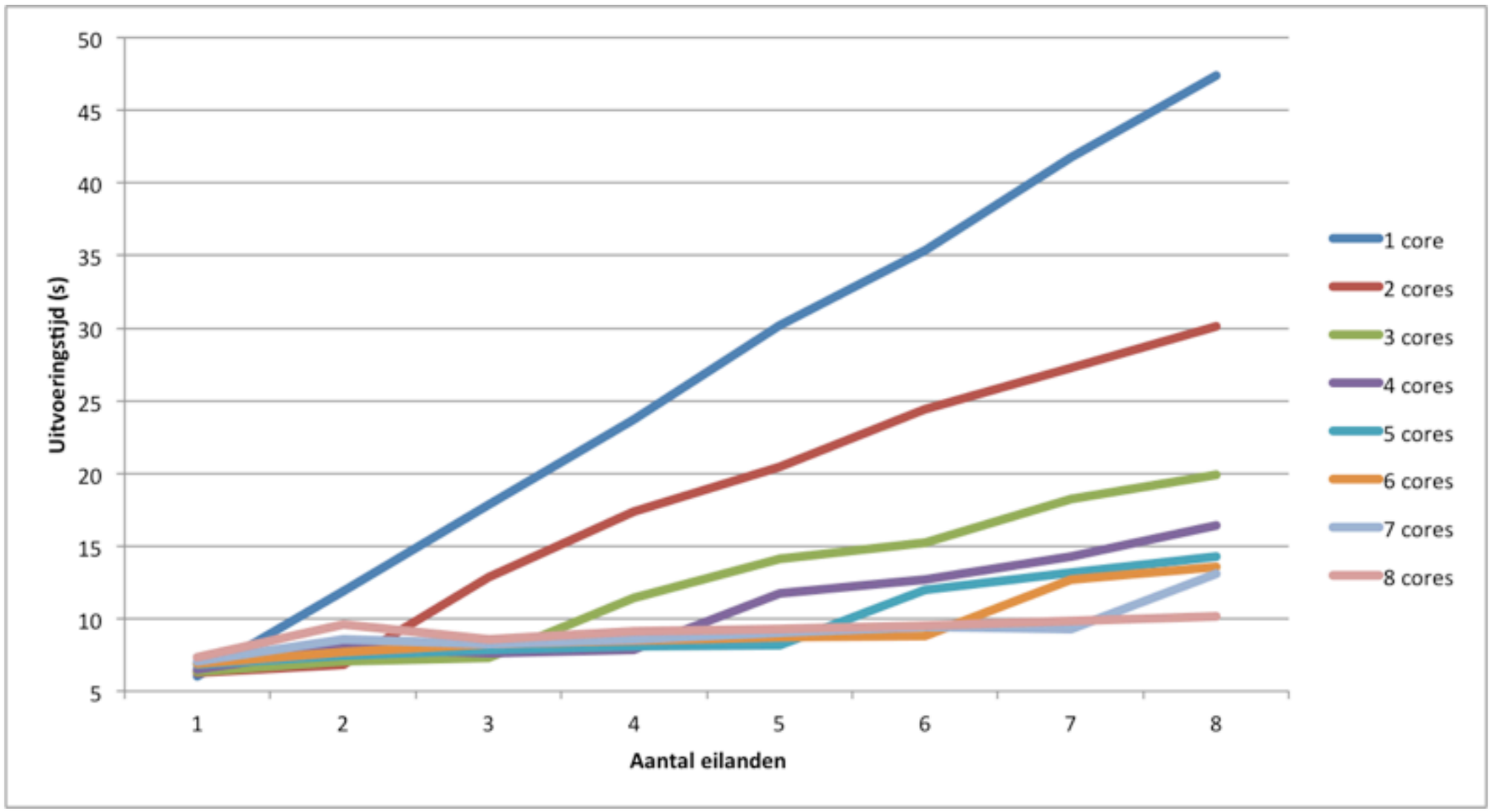
Linear increase in the number of islands (threads)



Framework performance

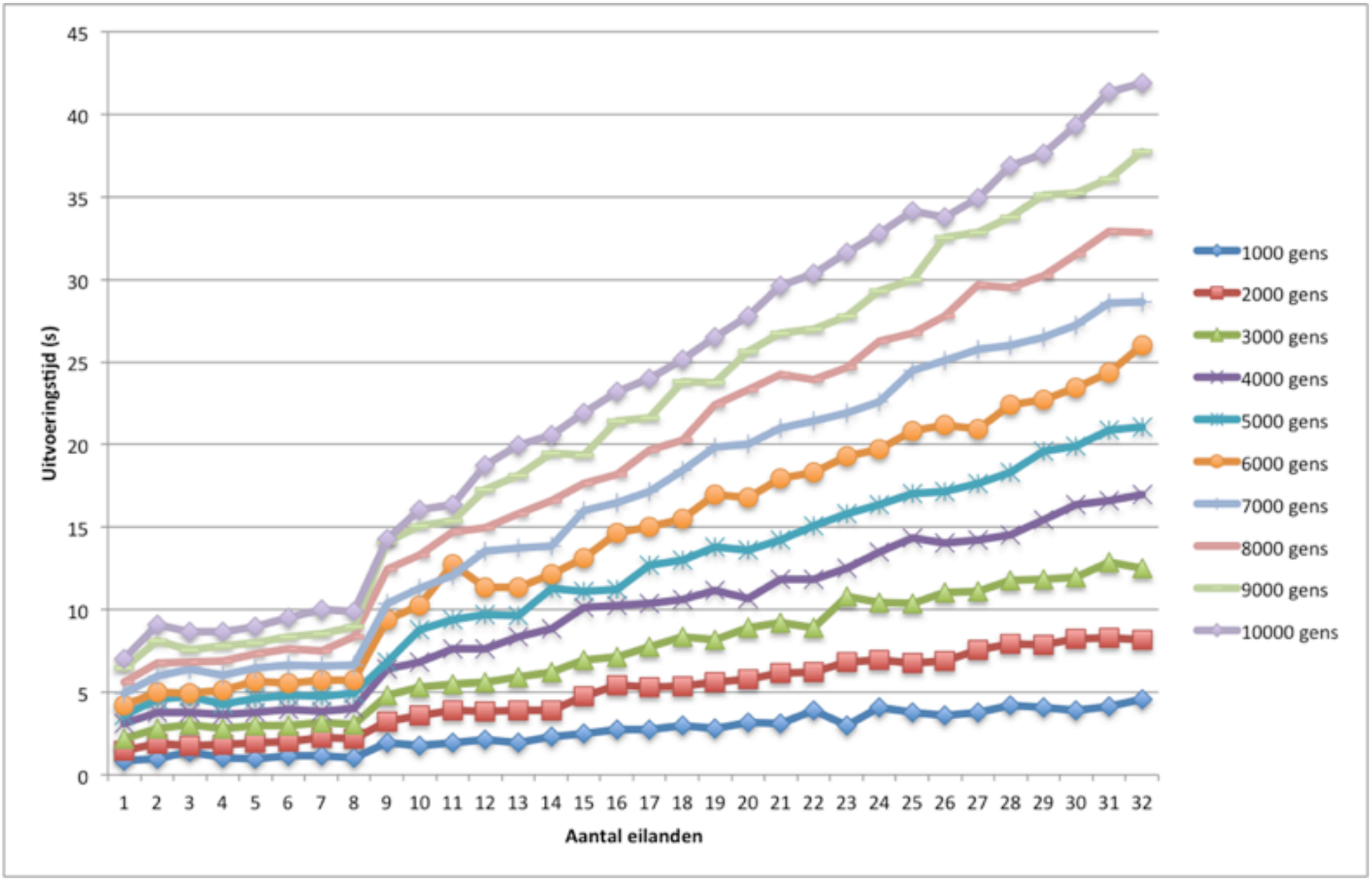
Impact of the number of islands

Linear increase when islands > cores



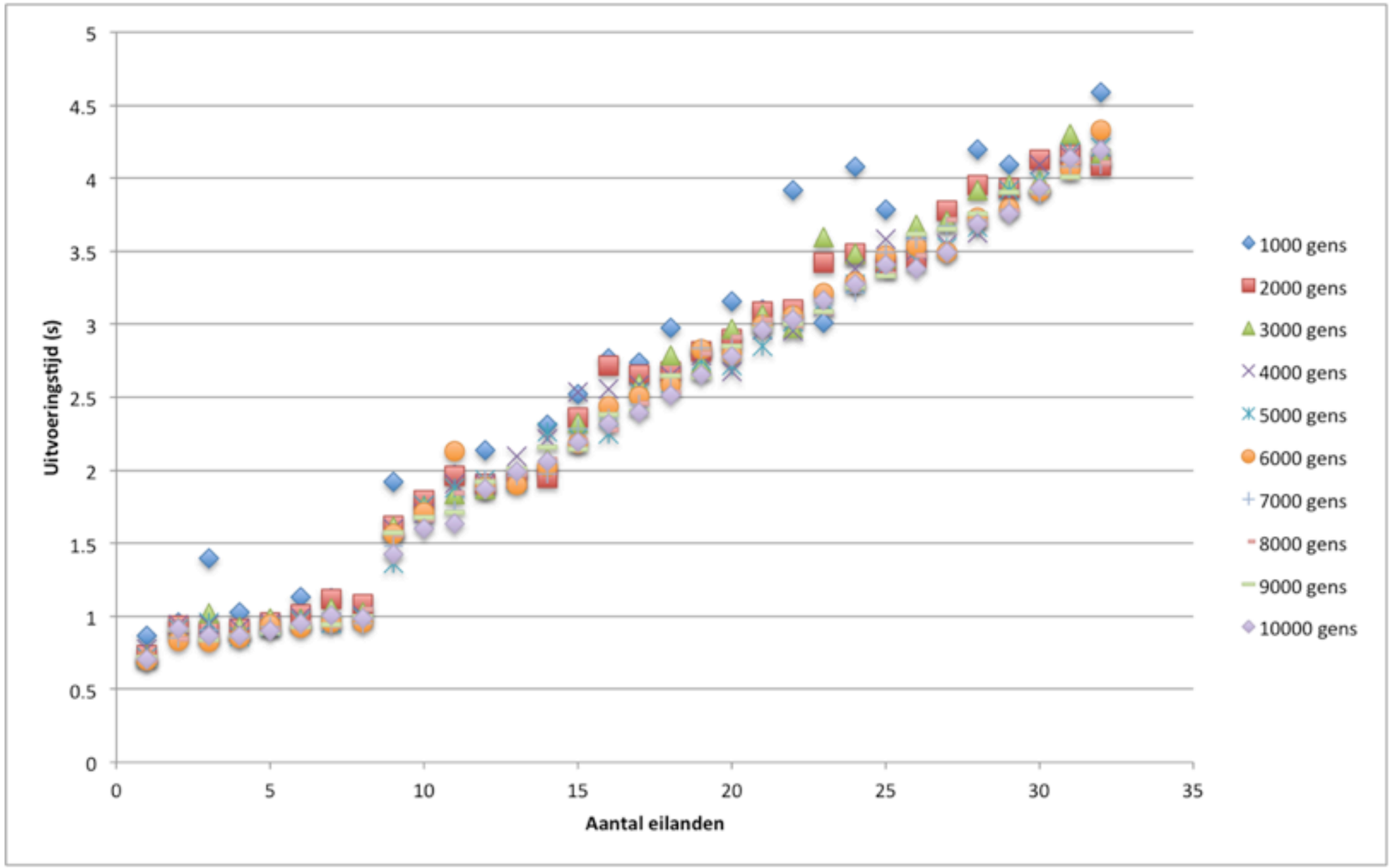
Framework performance

Impact of the number of generations



Framework performance

Impact of the number of generations



Conclusion

- The distributed method improves the fitness
  - The gain converges for a large amount of islands
  - In practice we will never use such large amounts
- Parallelism improves the execution time
  - If islands $\leq$ cores, there is a significant speed gain
  - Keep synchronization overhead in mind

Overview - Future work

1. Introduction
2. Parallelism
3. Framework
4. Parallelization
5. Evaluation
6. Future work

Future work

- Framework structure
  - Archives, checkpointing, selection
- Further experiments
  - Other benchmarks, more tests
- Parallel strategies
  - Reduce parallel overhead, asynchronous pollination
- Meta heuristics
  - Add other GA techniques (simulated annealing)

Questions?

Thank you for your attention!