Kindergarten Cop: Profiling-based Dynamic Nursery Resizing for GHC

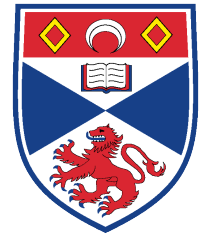
Henrique Ferreiro[1] Laura Castro[1] Vladimir Janjic[2]
David Castro[2] Kevin Hammond[2]

[1]University of A Coruña
Galicia

[2]University of St Andrews
Scotland

August 30, 2013

UNIVERSIDADE DA CORUÑA



University
of
St Andrews

Introduction

Motivated by my work on **profiling of parallel Haskell** programs, we have been looking at GHC's **garbage collector** (GC) performance.

Introduction

Motivated by my work on **profiling of parallel Haskell** programs, we have been looking at GHC's **garbage collector** (GC) performance.

Haskell gives us:

- very high level language features
- laziness
- parallelism as a library

Introduction

Motivated by my work on **profiling of parallel Haskell** programs, we have been looking at GHC's **garbage collector** (GC) performance.

Haskell gives us:

- very high level language features
- laziness
- parallelism as a library

This features rely on garbage collection, and...

Introduction

Motivated by my work on **profiling of parallel Haskell** programs, we have been looking at GHC's **garbage collector** (GC) performance.

Haskell gives us:

- very high level language features
- laziness
- parallelism as a library

This features rely on garbage collection, and...

Parallel performance relies on **parallel garbage collection performance**!

Basic vocabulary

- **Generational GC**: 2 generations, 2 steps in the first generation.
- Objects are allocated in the **allocation area** or **nursery**.
- The **mutator** is the actual program, independent of GC.

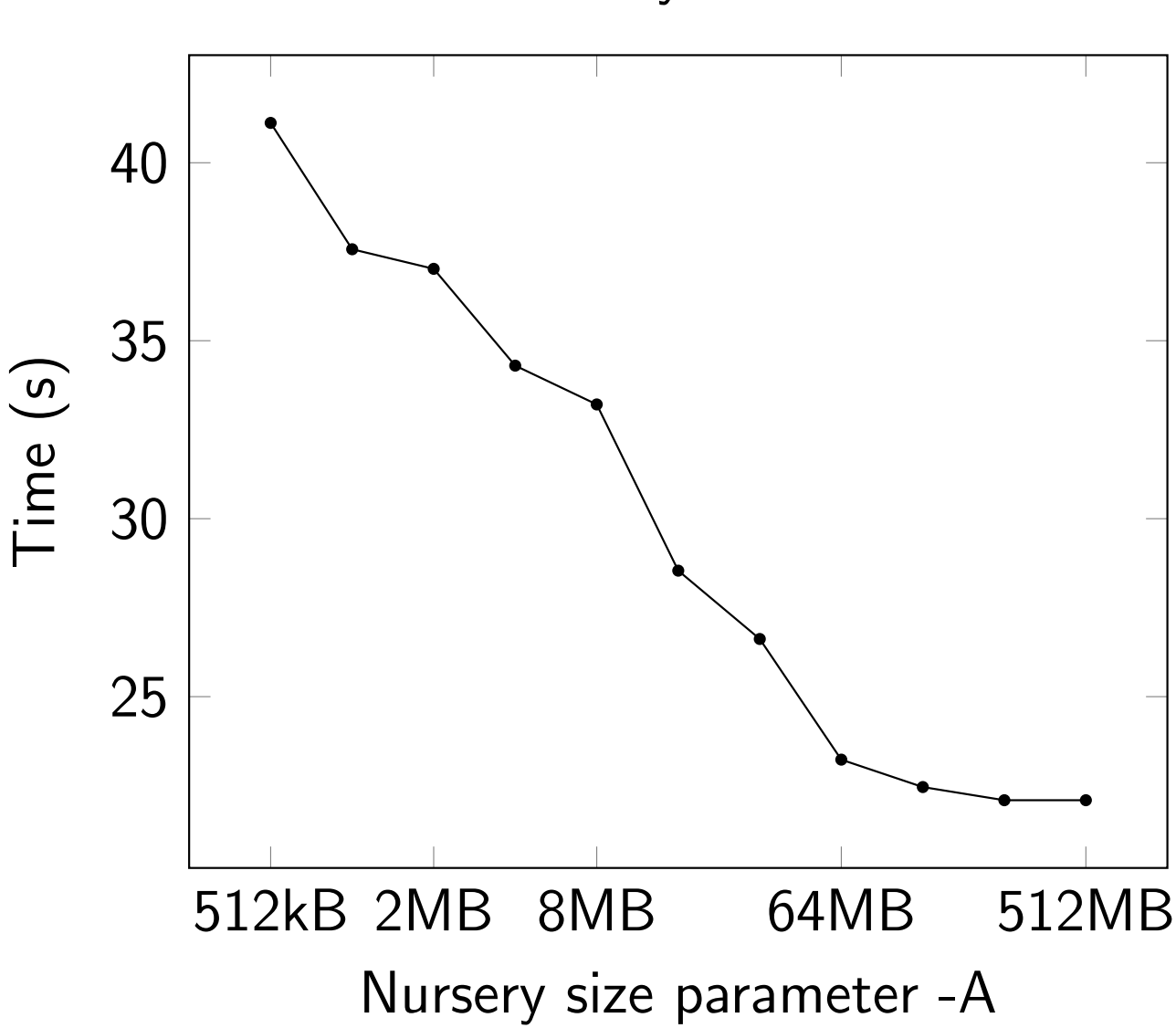
Motivation

GC performance can be a problem in functional programs.

```
./binary-trees +RTS -s
  19,439,366,976 bytes allocated in the heap
  21,905,868,680 bytes copied during GC
     127,729,448 bytes maximum residency (88 sample(s))
       2,127,288 bytes maximum slop
             368 MB total memory in use (0 MB lost due to fragmen

                                     Tot time (elapsed)  Avg pause
  Gen  0     36990 colls,     0 par   15.65s   15.68s     0.0004s
  Gen  1        88 colls,     0 par    9.08s    9.10s     0.1034s

  INIT    time    0.00s  (  0.00s elapsed)
  MUT     time   16.08s  ( 16.09s elapsed)
  GC      time   24.73s  ( 24.78s elapsed)
  EXIT    time    0.04s  (  0.04s elapsed)
  Total   time   40.85s  ( 40.91s elapsed)

  %GC     time      60.5%  (60.6% elapsed)
```

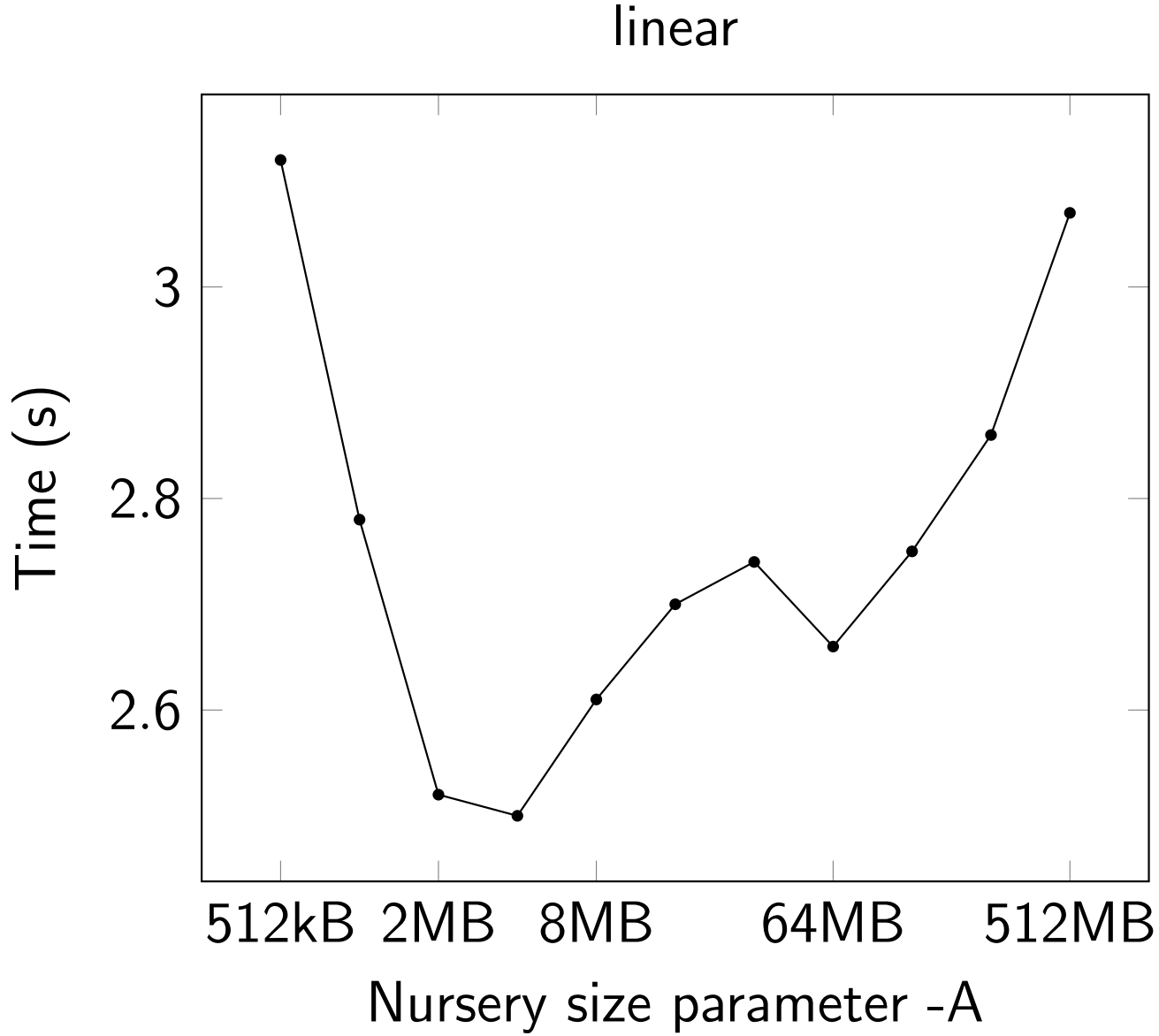
Motivation

Sometimes, it is fixed by **increasing the allocation area** size -A:



Motivation

Sometimes, this does not work:

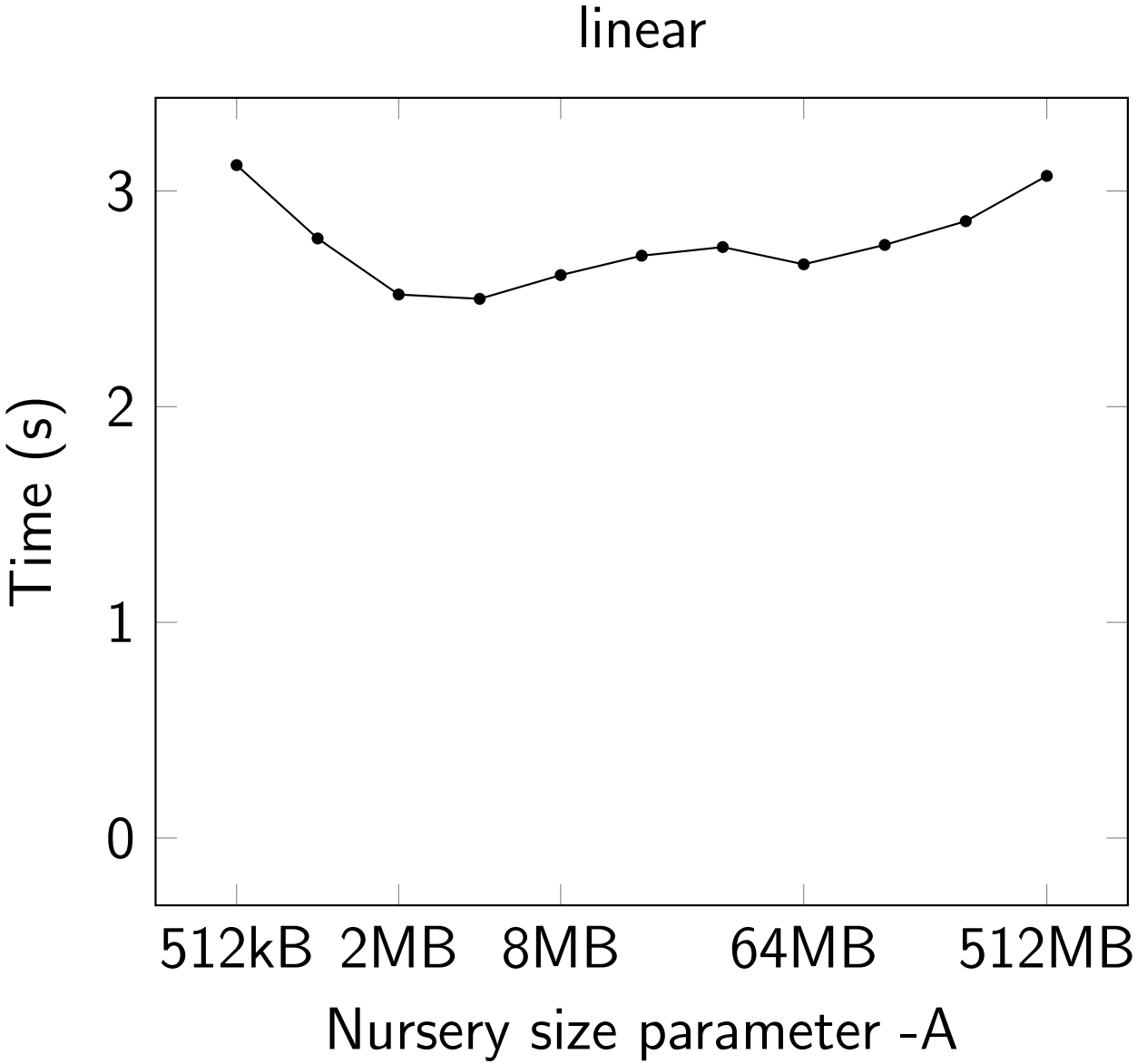


Motivation

Something else must be going on..

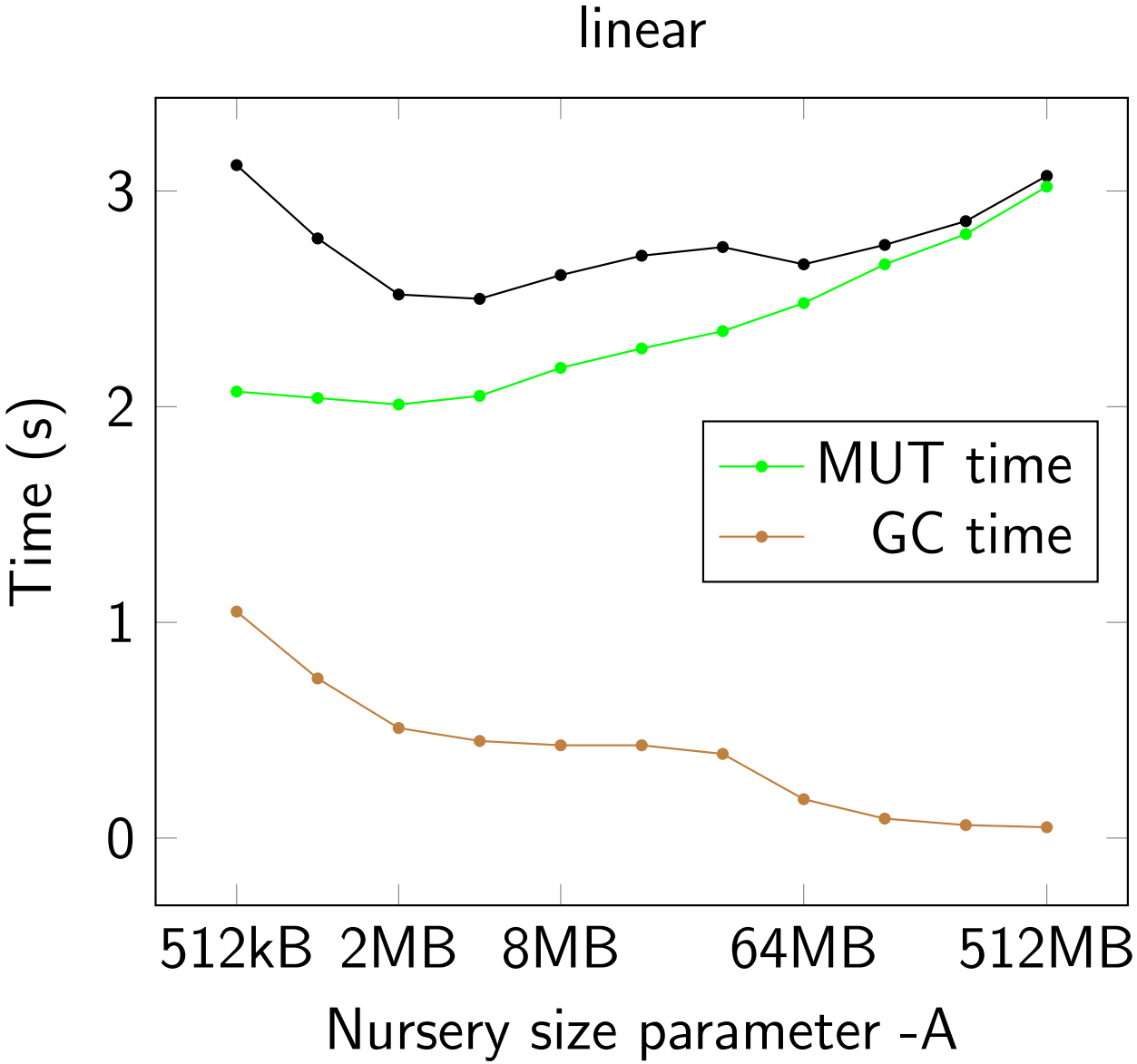
Motivation

Something else must be going on..



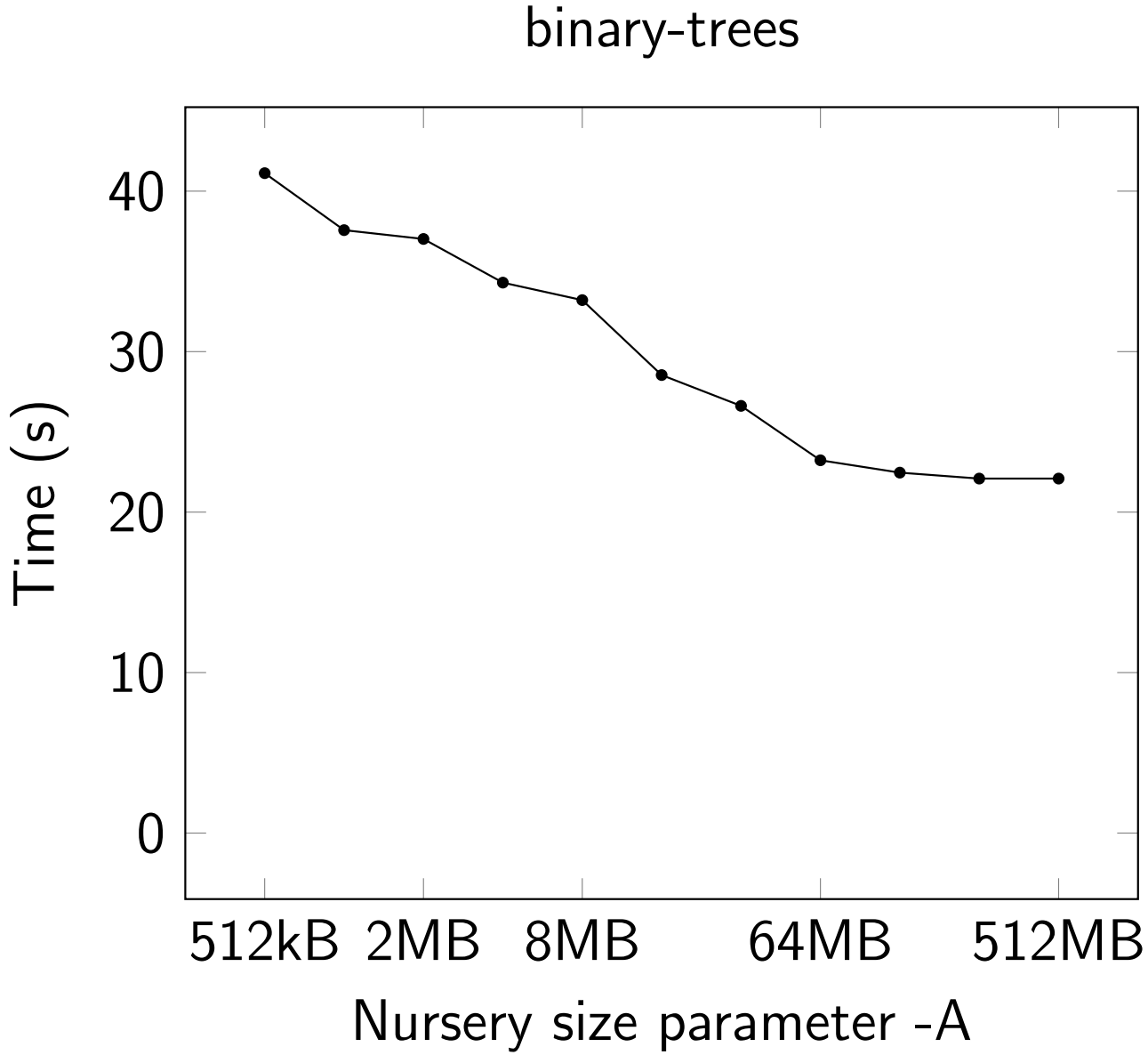
Motivation

Something else must be going on..



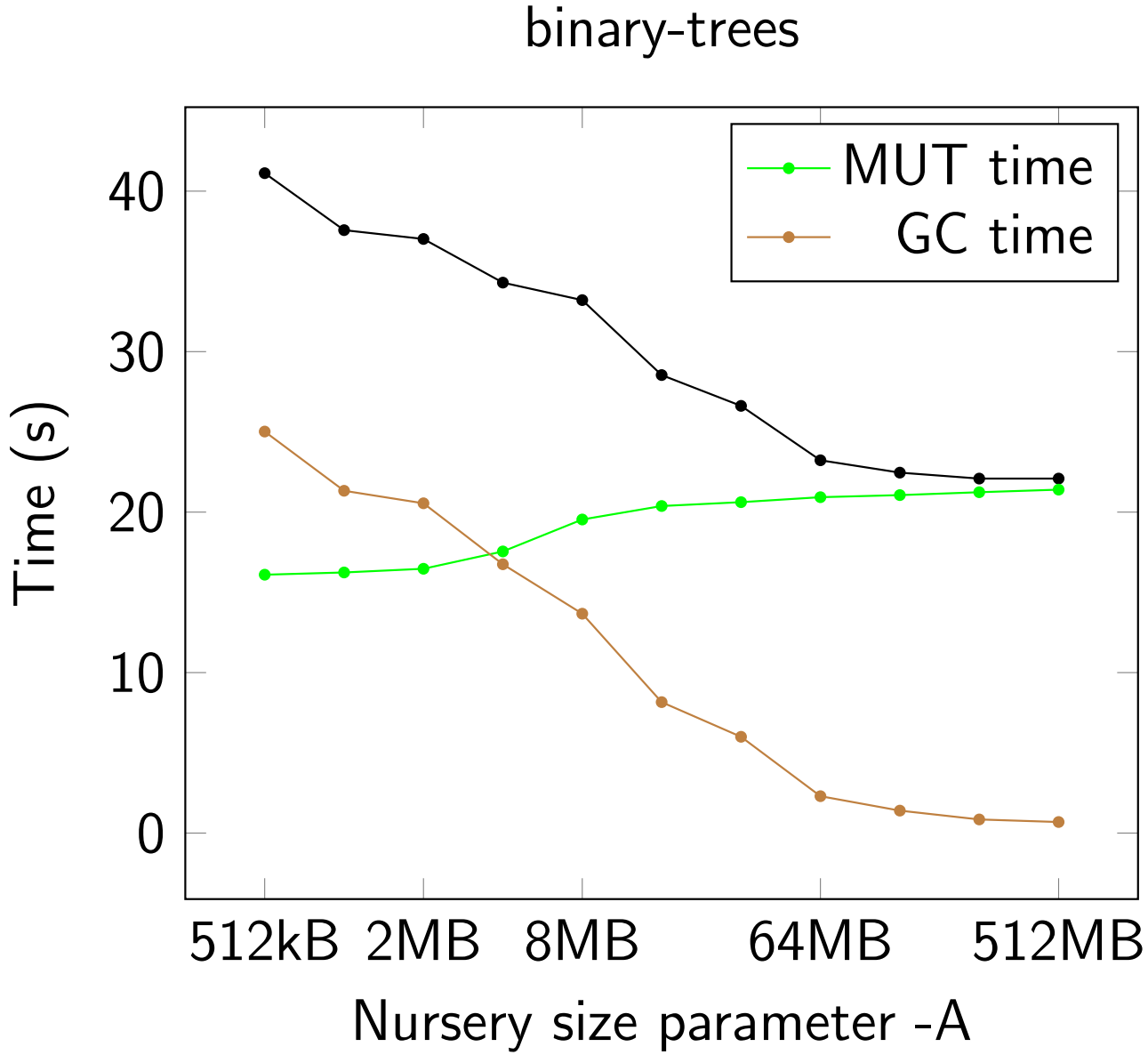
Motivation

Something else must be going on..



Motivation

Something else must be going on..



Our Work

Objectives:

- Develop a mechanism to decide on the **best nursery size** based on profiling information.
- Characterise the relation between the nursery size and the overall program performance.

Our Work

Objectives:

- Develop a mechanism to decide on the **best nursery size** based on profiling information.
- Characterise the relation between the nursery size and the overall program performance.

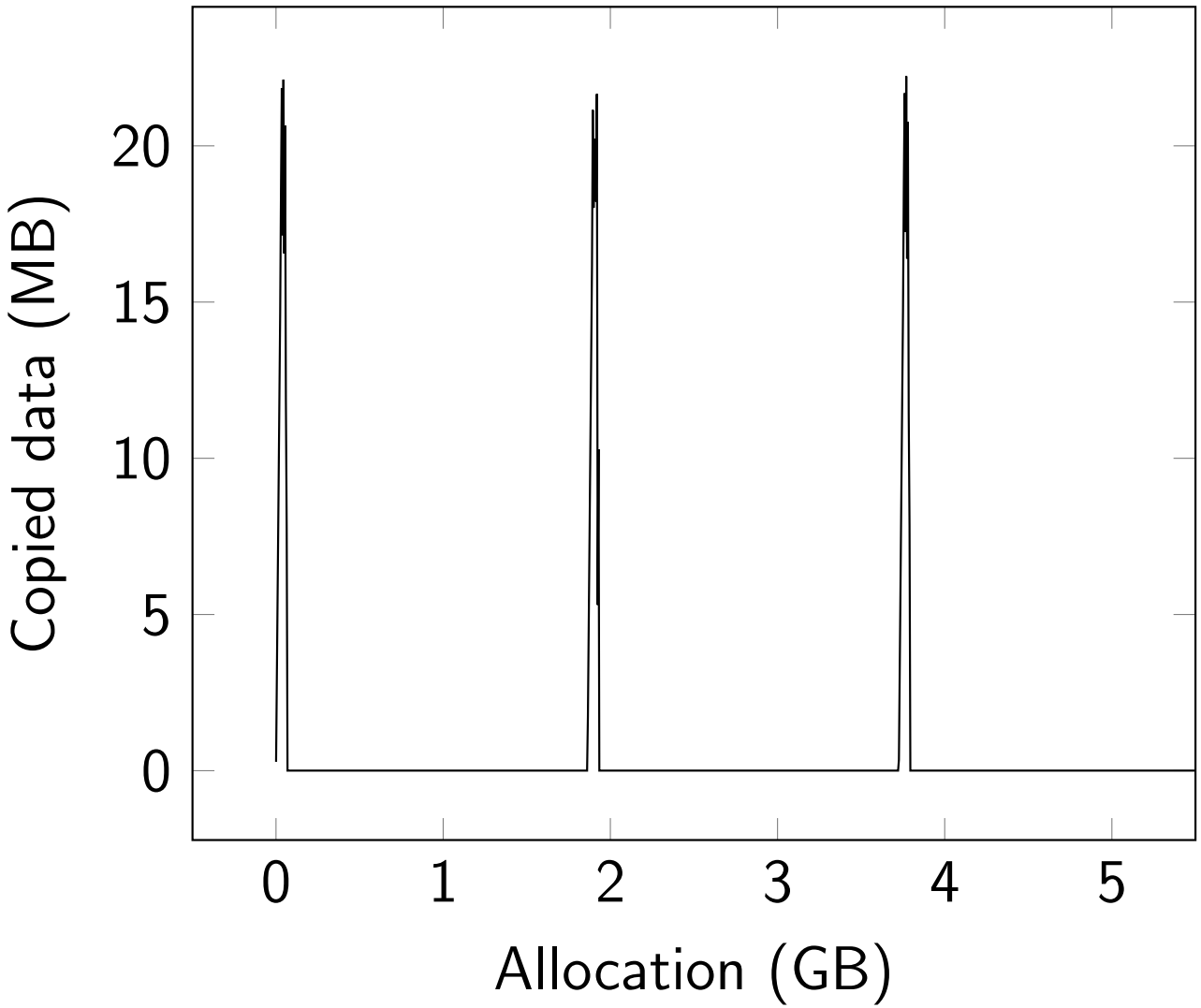
The result is some logic integrated into GHC's GC to dynamically adjust the allocation area size, so to **maximise performance of the total runtime**.

GC stress test

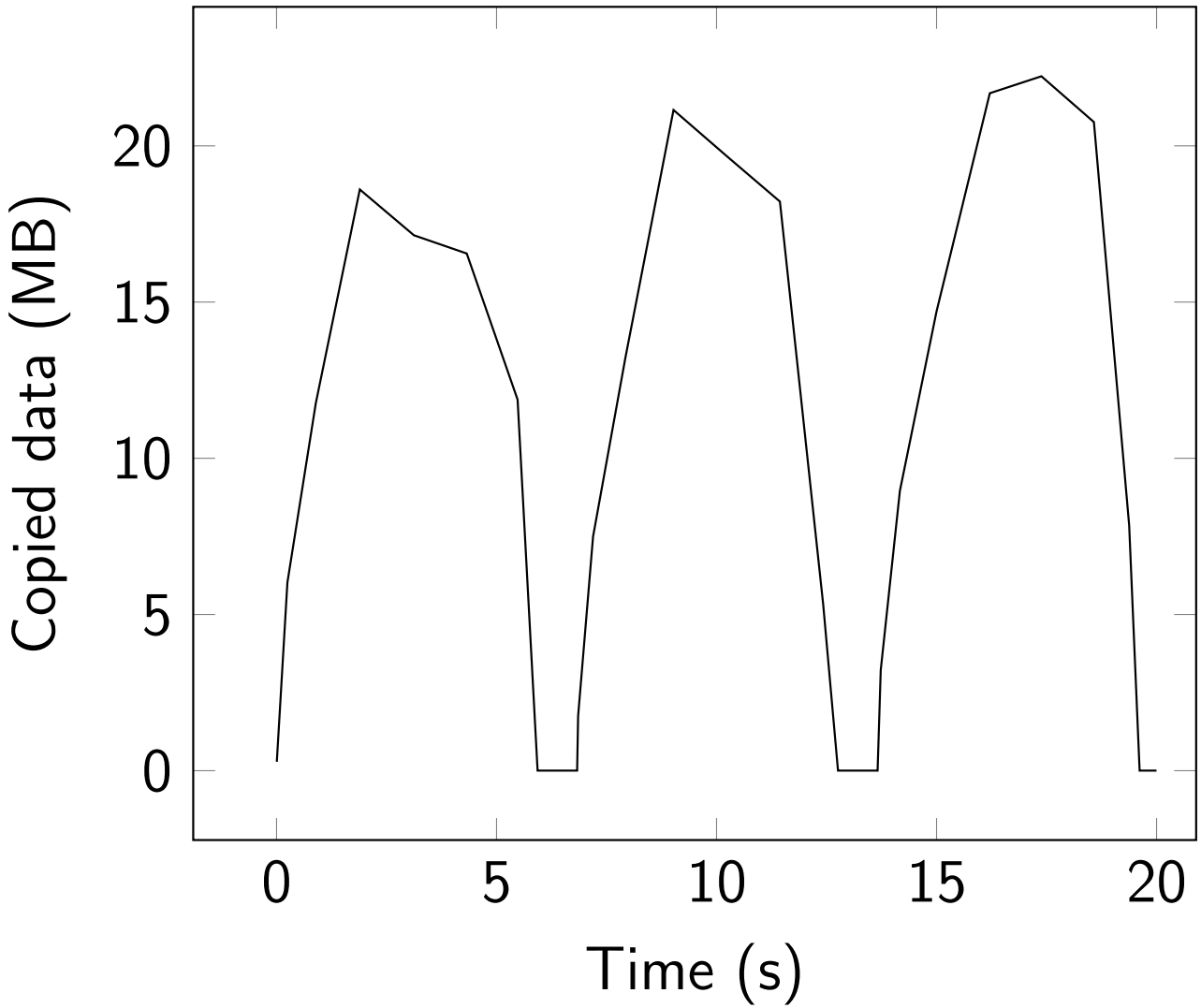
Very simple irregular example:

```
long :: Int -> Int -> Int
long n m = sum (foldl1 (zipWith' (+)) l)
  where
    l = take n (repeat [1..m])

f x = long 5 (mem*10000) + long (time*1000000) 1

main = print $ show $ take 10 (map f [1..])
```

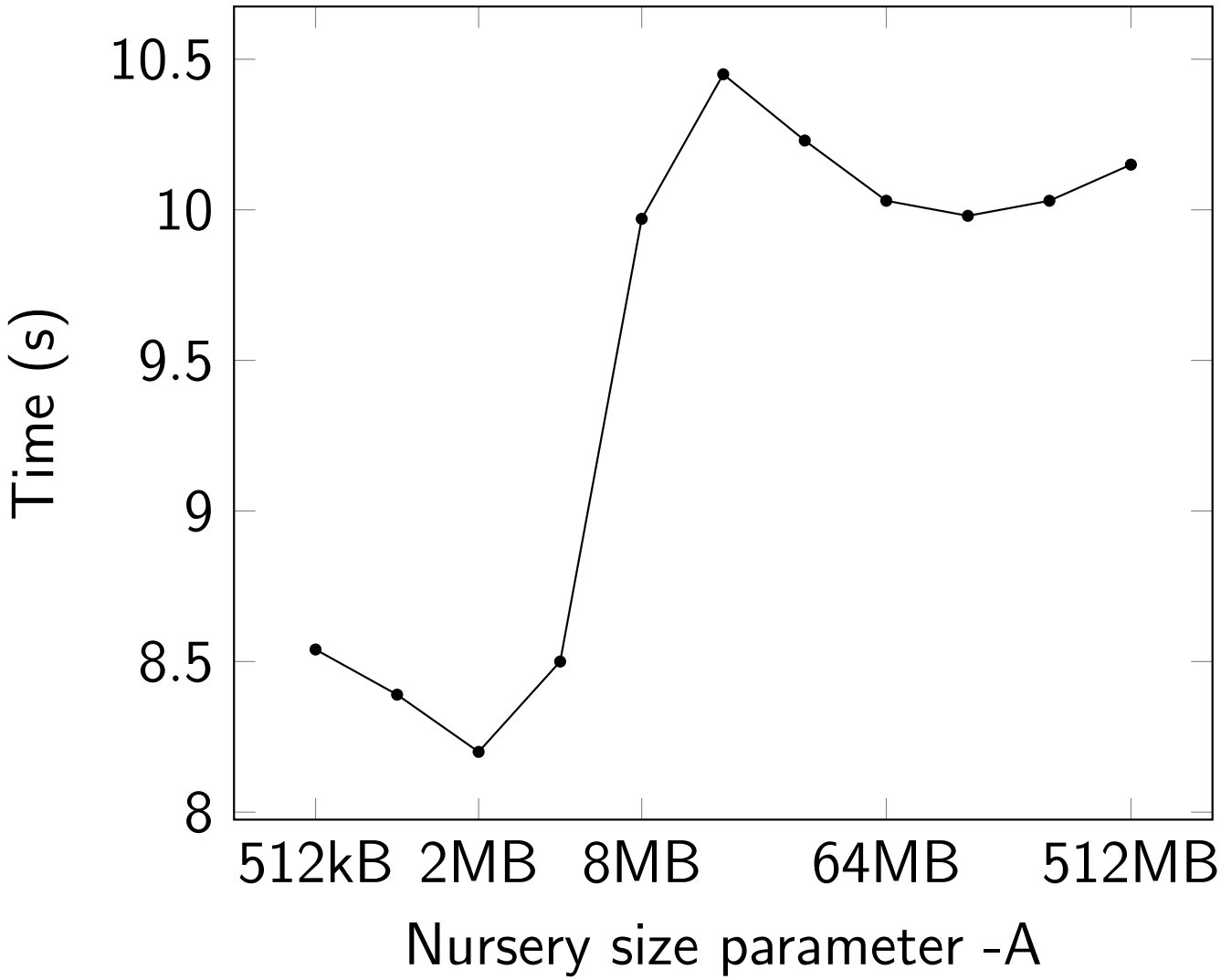
GC stress test



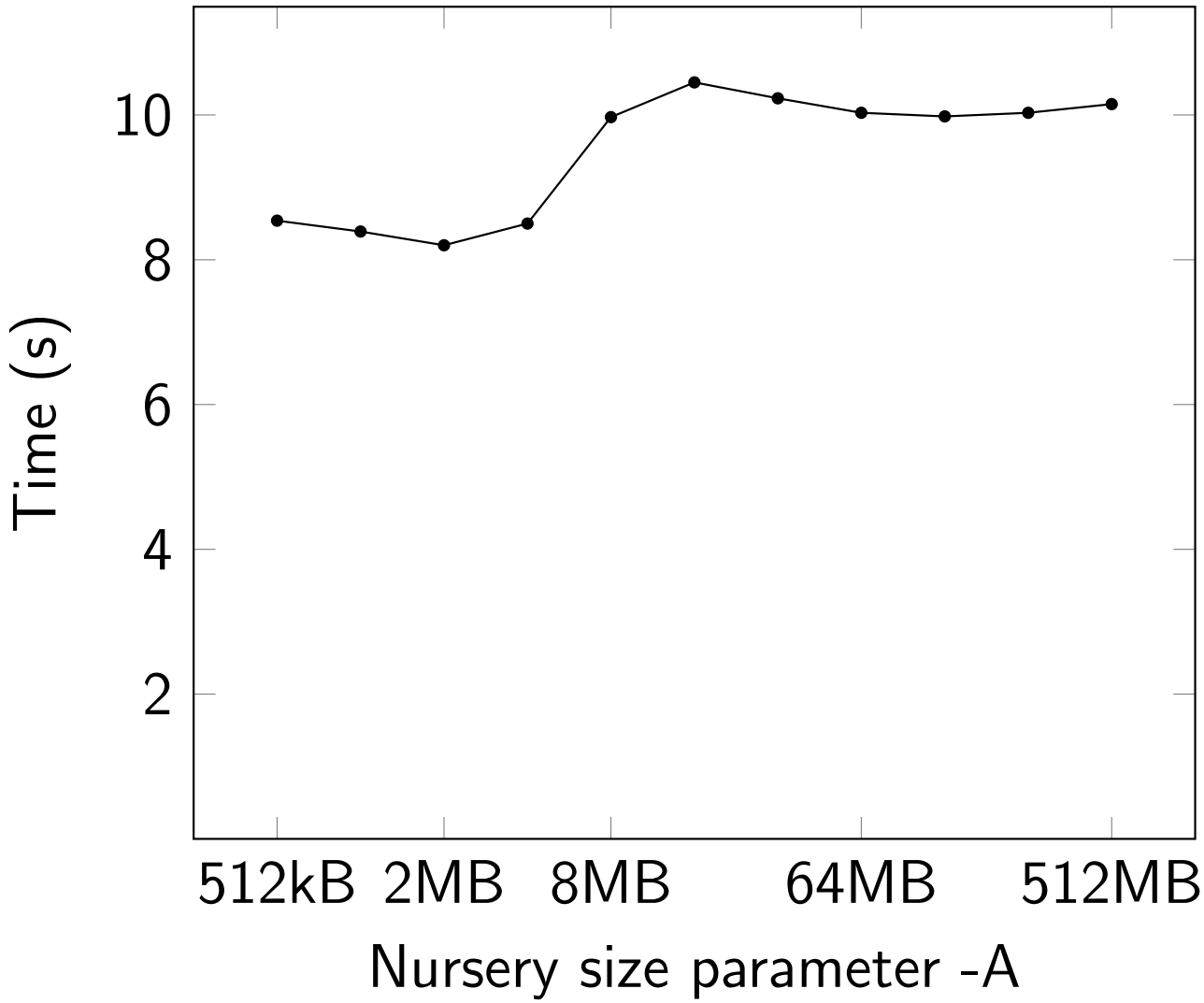
GC stress test



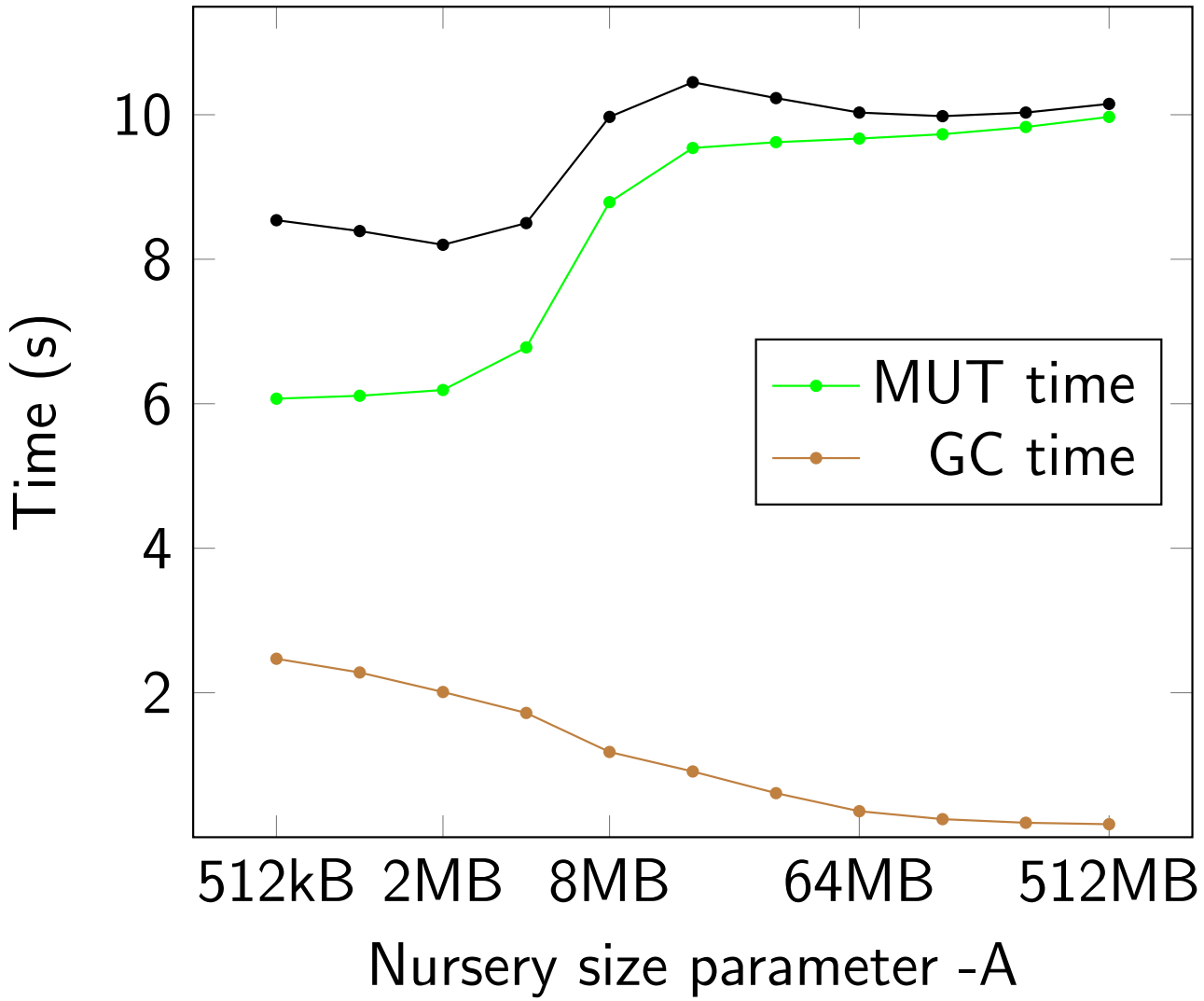
GC stress test



GC stress test



GC stress test



Analysis

From the previous plots, we can conclude that

- We don't know upfront what **choice of static nursery size** is the best.
- Different behaviour of MUT and GC time suggest a **dynamic approach**.

Analysis

From the previous plots, we can conclude that

- We don't know upfront what **choice of static nursery size** is the best.
- Different behaviour of MUT and GC time suggest a **dynamic approach**.

GHC's `-H` parameter gives a minimum heap size, and after substracting the needed amount for each next GC, leaves the **rest of the available memory** to the nursery.

Analysis

From the previous plots, we can conclude that

- We don't know upfront what **choice of static nursery size** is the best.
- Different behaviour of MUT and GC time suggest a **dynamic approach**.

GHC's `-H` parameter gives a minimum heap size, and after substracting the needed amount for each next GC, leaves the **rest of the available memory** to the nursery.

GHC's GC default heap growth strategy is not as good as other runtimes

GHC's ticket #3061 (`binary-trees`)

Profiling

Comparison of small allocation phases for -A2m and -A8m:

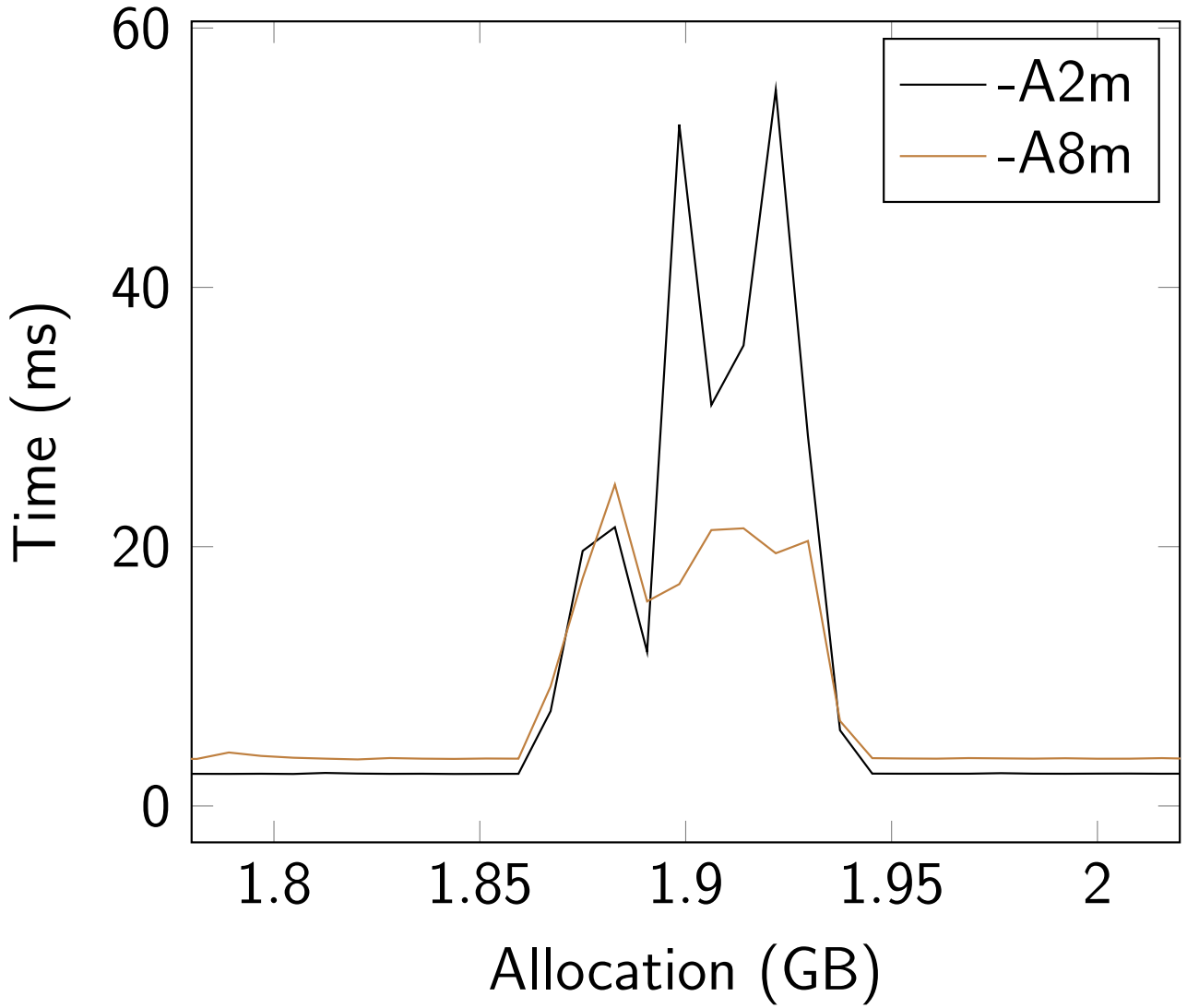
Nursery bytes	Copied bytes	GC elap	MUT elap	GC/alloc	MUT/alloc
...					
2097152	32	6817	610423	0.00325	0.2910
2097152	32	7155	618613	0.00341	0.2950
...					
2097152	56	6776	619085	0.00323	0.2952
2097152	56	6750	617927	0.00322	0.2947
...					
8388608	32	54832	3600299	0.00654	0.4292
8388608	32	56272	3579434	0.00671	0.4267
...					
8388608	56	55282	3554744	0.00659	0.4238
8388608	56	54532	3648542	0.00650	0.4349

Profiling

Comparison of big allocation phases for -A2m and -A8m:

Nursery bytes	Copied bytes	GC elap	MUT elap	GC/alloc	MUT/alloc
..					
2097152	1548304	2210379	838897	1.05399	0.40001
2097152	2078224	2533172	738132	1.20791	0.35197
...					
2097152	2890784	3215646	803685	1.53334	0.38322
2097152	4177920	4822656	1055495	2.29962	0.50310
...					
8388608	6576952	9349826	12064170	1.11459	1.43816
8388608	7583456	9902316	9587037	1.18045	1.14286
...					
8388608	6477792	11398443	4381135	1.35880	0.52227
8388608	7548016	9426002	7684111	1.12367	0.91602
...					

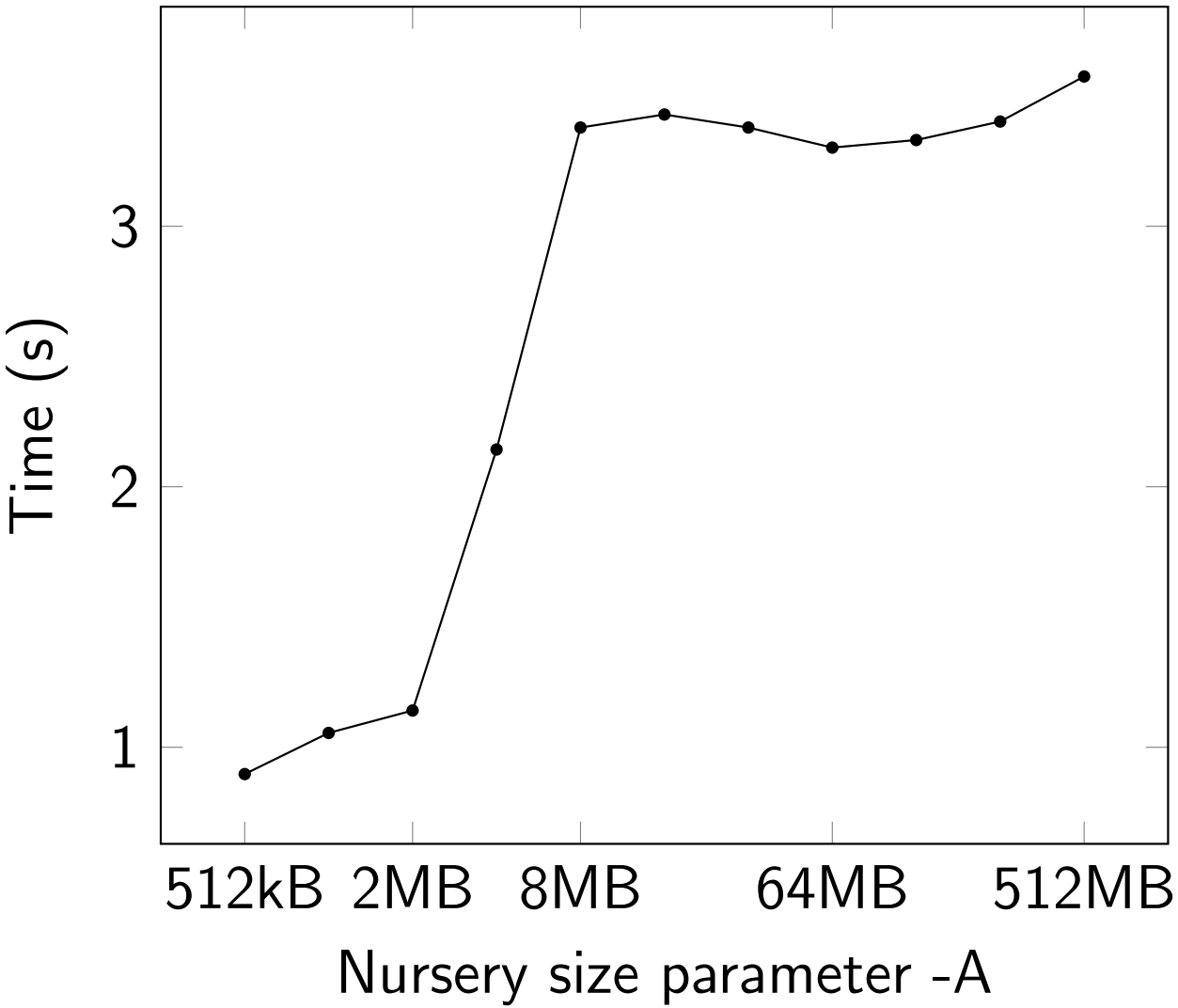
Profiling



Profiling

```
perf stat -e cache-references,cache-misses

 Performance counter stats for './Main +RTS -A2m':

         415050294 cache-references
           4734593 cache-misses              #    1,141 % of all

       8,201404216 seconds time elapsed

 Permance counter stats for './Main +RTS -A8m':

         385863220 cache-references
          13039380 cache-misses              #    3,379 % of all

      10,067264330 seconds time elapsed
```

Profiling

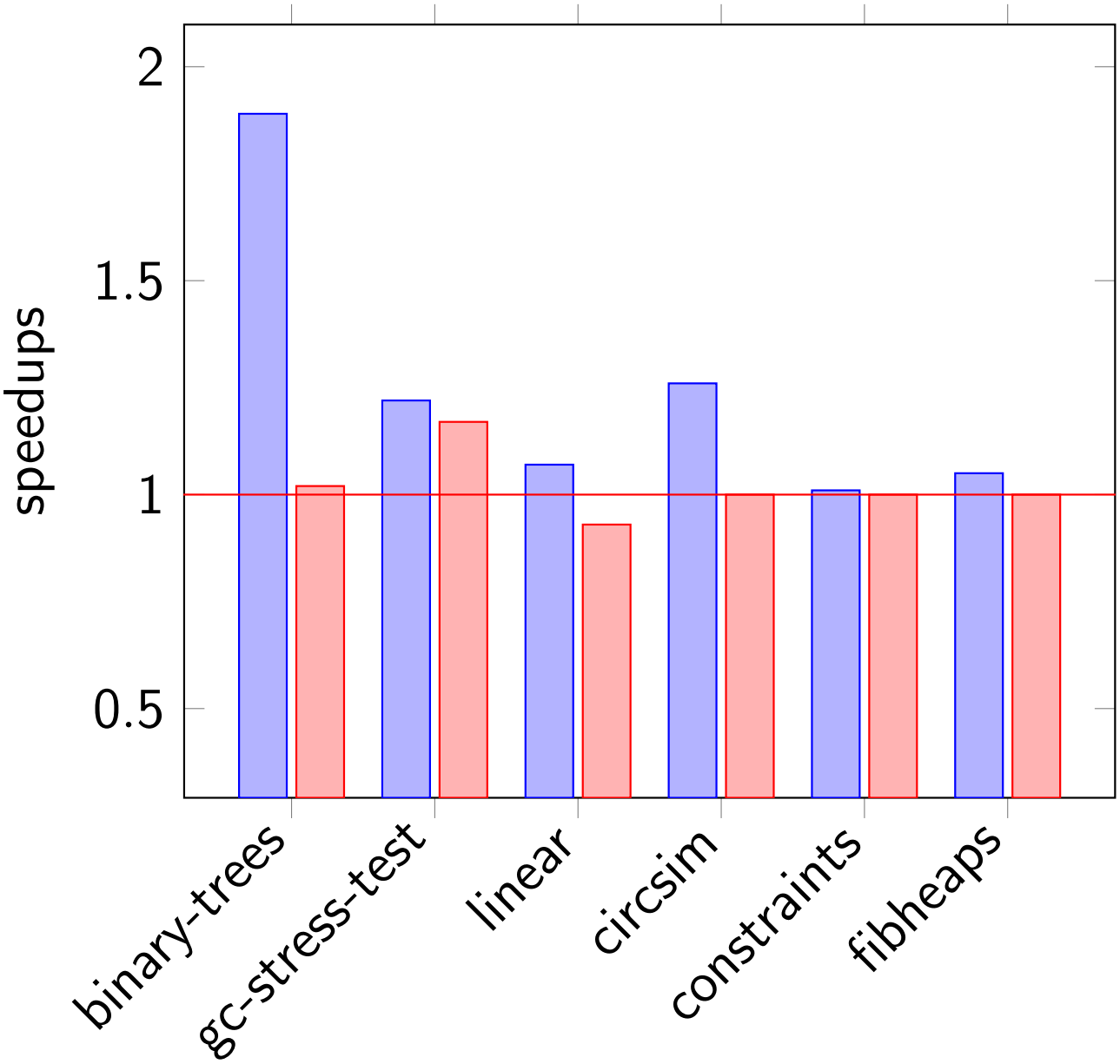


Early experiments

We have developed the necessary mechanism for GHC to make **dynamic adjustments of the nursery size** looking at past GC behaviour:

- Decide on a target **rate of copied data** on each GC.
- Increase/decrease the nursery to approach that rate.

Early experiments



Conclusion & Future Work

- Succesful early experiments
- Established relation between nursery size and performance

Conclusion & Future Work

- Succesful early experiments
- Established relation between nursery size and performance
- Use statistical analysis and/or control theory to learn from executions with different inputs.
- Validation of the technique.
- Application in a parallel setting.