GreenMalloc

Allocator Optimisation for Industrial Workloads

Aidan Dakhama W.B. Langdon Hector D. Menendez Karine Even-Mendoza
King's College London & University College London

Symposium on Search-Based Software Engineering (SSBSE)

The Problem: Allocators are Tricky

The Problem

Memory allocators (like GLIBC, TCMALLOC) have complex parameters.

Default settings are “one-size-fits-all” and are often inefficient for specific, complex, workloads.

The Impact

For long-running industrial workloads (e.g., gem5), this leads to:

- Wasted memory
- Slower performance
- Increased energy consumption

The Core Challenge

Challenge 1: Complexity

The parameter search space is high-dimensional, mixed discrete–continuous. Manual tuning is difficult.

Challenge 2: Speed

Optimising directly on gem5 is impractical – a single evaluation run can take hours or even days.

Our Solution: GreenMalloc

> “ *A framework to automatically find energy- and memory-efficient allocator configurations using a fast proxy benchmark.* ”

The Proxy: RAND_MALLOC

A lightweight proxy benchmark to explore allocator parameters efficiently.

Trace System

Generate a seed trace from the real system (gem5).

Generate Proxy

RAND_MALLOC creates a fast, synthetic workload.

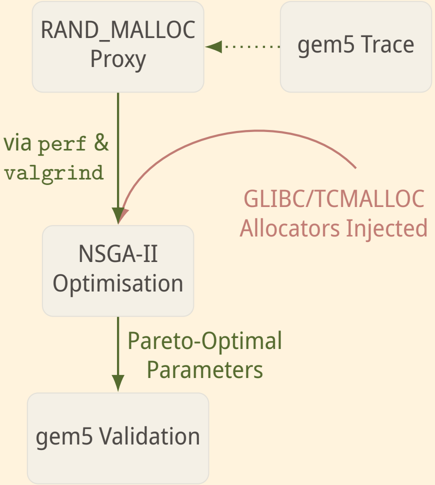
Emulate Behaviour

Exercises memory allocation patterns, acting as a proxy for the full system.

This avoids the prohibitive cost of optimising directly on gem5, which can take hours or days per run.

The GreenMalloc Workflow

- **Explore:** Use a genetic algorithm to optimise over the fast RAND_MALLOC proxy.
- **Evaluate:** Optimise for green and performant characteristics – Peak Memory, and Instructions.
- **Transfer:** Select the pareto-optimal solutions, and validate them against gem5.
- **Validate:** Keep the best transferred configurations on the real gem5 system to get final results



The Search Problem

Algorithm

NSGA-II
Population: 24
Generations: 500

Objective 1: Green

Minimise Peak Heap Usage
(Measured by `valgrind`)

Objective 2: Performance

Minimise Instruction Count
(Measured by `perf`)

Experimental Setup

We tested the transferability with 50 C programs on gem5's System Emulation (SE) mode, comparing four configurations. We also repeated each search 5 times, across all configurations.

Allocator	Tuning	Parameter Count	Search Space
GLIBC	Default	N/A	N/A
GLIBC	**Tuned**	**7**	$\approx \mathbf{7 \times 10^{35}}$
TCMALLOC	Default	N/A	N/A
TCMALLOC	**Tuned**	**13**	$\approx \mathbf{2 \times 10^{41}}$

RQ1: Did the Search Find Trade-Offs?

Yes. The search successfully identified different trade-off profiles for each allocator.

GLIBC

- Showed **more gradual trade-offs** (slope: -0.216).
- Produced larger Pareto fronts (avg. 3 solutions).
- This suggests its default settings have more room for optimisation.

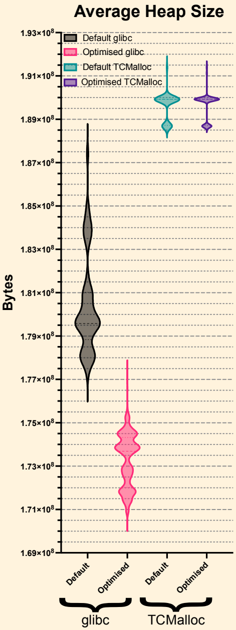
TCMALLOC

- Had a **much steeper trade-off** (slope: -3.17).
- Produced smaller fronts (avg. 1.6 solutions).
- This suggests it operates closer to its optimal boundaries, requiring more aggressive trade-offs.

RQ2: Results on gem5 (Memory)

Average Heap Size

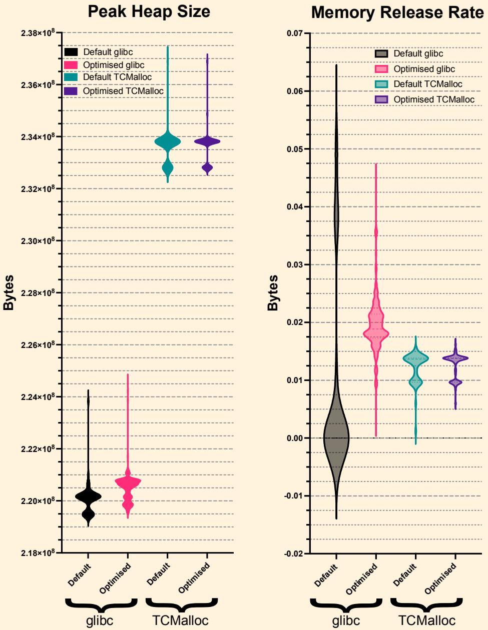
- **GLIBC:** Showed clear improvement.
- Mean reduced by ≈4% (180.4M to 173.3M bytes).
- Tighter standard deviation, indicating more stability.
- **TCMALLOC:** Mean was nearly unchanged.
- Showed less variance, improving predictability.



RQ2: Results on gem5 (Memory)

Peak Heap & Release Rate

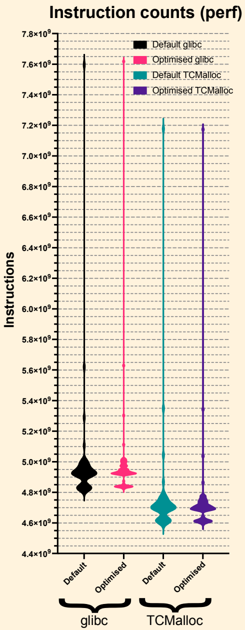
- **Peak Heap:** No significant reductions for either allocator.
- This suggests peak usage was already near minimum.
- **Memory Release Rate:**
- **GLIBC:** Benefitted significantly. Free rate rose from 0.0080 to 0.0196 (≈2.4x faster).
- **TCMALLOC:** Also improved, with a higher average release rate.



RQ2: Results on gem5 (Performance)

Instruction Counts

- Both allocators saw small but measurable reductions in instructions.
- **GLIBC:** Reduced mean instructions from 4.992×10^9 to 4.990×10^9, with tightened deviation.
- **TCMALLOC:** Reduction in mean instructions from 4.77×10^9 to 4.76×10^9, with less variance.



RQ2: Results on gem5 (Best Case)

Conclusion

- GreenMalloc found one TCMALLOC configuration that was a clear "win-win".
- It achieved a **4.65% reduction** in instruction count...
- ...and a **2.06% reduction** in peak heap usage at the same time.
- This shows the potential of the approach.

Conclusion & Future Work

Conclusion

- We introduced GreenMalloc, a search-based framework for allocator tuning.
- We utilise a lightweight proxy to efficiently optimise complex systems.
- We show gains for both GLIBC and TCMALLOC on gem5.

Future Work

- Apply this strategy to other aspects of complex software.
- Target gem5's full system (FS) mode.
- Explore broader targets, including:
  - Virtual Machines (VMs)
  - Containerisation Systems
  - Other Simulators

Thank You

Check out our tool below

Aidan Dakhama
aidan.dakhama@kcl.ac.uk