Optimizing Layout of Recursive Datatypes with Marmoset (Artifact)

Vidush Singhal
Purdue University, West Lafayette, IN, USA

Chaitanya Koparkar
Indiana University, Bloomington, IN, USA

Joseph Zullo
Purdue University, West Lafayette, IN, USA

Artem Pelenitsyn
Purdue University, West Lafayette, IN, USA

Michael Vollmer
University of Kent, UK

Mike Rainey
Carnegie Mellon University, Pittsburgh, PA, USA

Ryan Newton
Purdue University, West Lafayette, IN, USA

Milind Kulkarni
Purdue University, West Lafayette, IN, USA

Abstract

While programmers know that memory representation of data structures can have significant effects on performance, compiler support to *optimize* the layout of those structures is an under-explored field. Prior work has optimized the layout of individual, *non-recursive* structures without considering how collections of those objects in linked or *recursive* data structures are laid out.

This work introduces Marmoset, a compiler that optimizes the layouts of algebraic datatypes, with a special focus on producing highly optimized, *packed* data layouts where recursive structures can be traversed with minimal pointer chasing. Marmoset performs an analysis of how a recursive ADT is used across functions to choose a *global* layout that promotes simple, strided access for that ADT in memory. It does so by building and solving a constraint system to minimize an abstract cost model, yielding a predicted efficient layout for the ADT. Marmoset then builds on top of Gibbon, a prior compiler for packed, mostly-serial representations, to synthesize optimized ADTs. We show experimentally that Marmoset is able to choose optimal layouts across a series of microbenchmarks and case studies, outperforming both Gibbon's baseline approach, as well as MLton, a Standard ML compiler that uses traditional pointer-heavy representations.

2012 ACM Subject Classification Software and its engineering → Compilers; Software and its engineering → Software performance; Information systems → Data layout

Keywords and phrases Tree traversals, Compilers, Data layout optimization, Dense data layout

Digital Object Identifier 10.4230/DARTS.10.2.21

Acknowledgements This work was supported in part by NSF CCF-1908504, CCF-1919197, CCF-2216978, CCF-2119352, CCF-1909862 and EPSRC EP/X021173/1.

Related Article Vidush Singhal, Chaitanya Koparkar, Joseph Zullo, Artem Pelenitsyn, Michael Vollmer, Mike Rainey, Ryan Newton, and Milind Kulkarni, "Optimizing Layout of Recursive Datatypes with Marmoset: Or, Algorithms + Data Layouts = Efficient Programs", in 38th European Conference on Object-Oriented Programming (ECOOP 2024), LIPIcs, Vol. 313, pp. 38:1–38:28, 2024.
`https://doi.org/10.4230/LIPIcs.ECOOP.2024.38`

Related Conference 38th European Conference on Object-Oriented Programming (ECOOP 2024), September 16–20, 2024, Vienna, Austria

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2024 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

This artifact is an accompaniment to the conference paper "Optimizing Layout of Recursive Datatypes with Marmoset". This artifact helps validate the claims made in the evaluation section of the paper. It helps to validate the results shown in Table 1-5 and Figures 8 and 9.

2 Content

The artifact package includes:

- A README with instructions on how to install the artifact.
- A Dockerfile to build a Docker [3] image.
- A tar ball of the pre built docker image.
- Python scripts to run the experiments.
- Source files of benchmarks.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at:

- Zenodo: `https://zenodo.org/records/10732462`
- Github: `https://github.com/iu-parfunc/ecoop24-marmoset-artifact.git`

For the most current version of the artifact, please check the github repository for the most recent updates.

4 Tested platforms

The Artifact will work on any CPU configuration. However, it requires a large RAM for bigger datasets to work. The experiments were run on an AMD Ryzen Threadripper 3990X 64-Core processor. It has 252 Gb of RAM and the operating system is Ubuntu 22.04.3 LTS. It is recommended to run the experiments on a system with $>$ 100 GB of RAM.

5 License

The artifact is available under The Creative Commons Attribution 4.0 International License. [1]

6 MD5 sum of the artifact

224d1336264a5fd07c2cc56d525ee33e

7 Size of the artifact

3.0GB

8 ECOOP 2024 Artifact – User Guide

Title of the submitted paper:

Optimizing Layout of Recursive Data Types with Marmoset

8.1 Quick-start guide

The artifact is bundled as an OCI container created with Podman (Dockerfile is available). The Docker image is tarred as marmosetArtifact.tar The image can be added to the local Docker store as follows:

```
docker load --input marmosetArtifact.tar
```

Alternatively, you can build the image with the Dockerfile

```
DOCKER_BUILDKIT=1 docker image build -t marmoset -f Dockerfile .
```

Once you get the image, start the session as follows (so called CMD1):

```
docker run --rm -ti marmoset
```

For the kick-the-tires stage, execute the following commands upon entering the container:

```
cd vsGibbon
./generate_runtimes.py --run quick
```

This should take about a minute and at the end print the contents equivalent to the contents of Tables 1 and 2 in the paper. The quick mode is specifically developed for the kick-the-tires stage.

8.2 Overview

- type of artifact – code, in particular:
  - source files of the Gibbon [6, 5] compiler with our extension to it called Marmoset.
  - Gibbon binaries including Marmoset.
  - Python and Bash scripts to run Gibbon, Marmoset and GHC to reproduce the main tables and figures of the paper.
- format – Marmoset is implemented in Haskell (like the rest of Gibbon). The scripts produce a combination of .csv and .pdf files holding the data in the tables and figures. Also, the benchmark programs discussed in the paper are stored as files with the .hs extension.
- location in the container – After entering the container via the instructions provided above, the structure of the container is as follows. The default working directory and the $HOME in the container is /root. The run.sh script in this directory may serve as a master script to run either the small set of inputs or the full set (the default). The two folders ~/vsGibbon, ~/vsGHC and ~/vsSML contain benchmarks to evaluate Marmoset against Gibbon, GHC and standard ML, following the Evaluation section of the paper. The ~/marmoset directory contains the Gibbon compiler with the Marmoset extension. The code is pre-built and available in $PATH as gibbon (Marmoset is activated by gibbon flags).

8.3 Proof of availability

We posted the artifact on Zenodo in addition to Dagstuhl: https://doi.org/10.5281/zenodo.10578861. The artifact is available under the Creative Commons Attribution 4.0 International license.

8.4 Proof for a functional or reusable badge

8.4.1 Files structure in the container

The source files of Gibbon and Marmoset reside in `~/marmoset` (`~` is `/root`).

All scripts and benchmarks reside in one of the three directories in the container:

1. `~/vsGibbon` – evaluation for Gibbon and Marmoset (Tables 1–3 and Figure 9).
2. `~/vsSML` – evaluation for Marmoset vs MLton (Figure 10).
3. `~/vsGHC` – evaluation for Marmoset vs GHC (for the extended version of the paper).

In (1) and (3), there are two subdirectories that contain `small` and `large` benchmarks respectively. The two kinds of benchmark programs differ only in sizes of inputs. The `large` variant should more faithfully reproduce the results in the paper but requires big RAM (>100Gb).

Four Python scripts map on the figures and tables in the paper as follows:

1. `~/vsGibbon/generate_runtimes.py` – generates the run times for Gibbon, Marmoset-greedy, and Marmoset-solver, `Tables 1-3`
2. `~/vsGibbon/generate_compile_times.py` – generates the compile times, `Figure 9`.
3. `~/vsGibbon/generate_cache_stats.py` – generates the statistics for CPU cache, `Table 5`.
4. `~/vsSML/generate_sml_numbers.py` – generates the run times for MLton, `Figure 8`.
5. `~/vsGHC/generate_ghc_numbers.py` – generates the run times for GHC as presented in the extended version of the paper [4].

Script (3) relies on the PAPI [2] framework, which does not work inside a container. Below we provide instructions for running it outside the container (section "Build Marmoset and PAPI outside Docker for generating Table 5").

The other three scripts can run sequentially from one master script called `~/run.sh`. The master script accepts the `small` flag, as well as scripts (1) and (5), so that the results can reproduce at a smaller scale on an average consumer machine using the benchmarks in the `small` directories.

After executing `CMD1` (see above) and entering the container, use either the master script `~/run.sh` or the individual scripts to reproduce the figures and tables. For example, `./generate_runtimes.py` (inside `vsGibbon`) generates the run times for Gibbon and Marmoset, prints them in a tabular form, and stores them in CSV files. In addition to the raw run times, the script also prints out the speedups in a separate column (the last column). In particular,

`speedup` = (slowest Gibbon, i.e, red color in the table) / `Marmoset_solver`.

8.4.2 Approximate timings

1. `~/vsGibbon/generate_runtimes.py`

   - `small` mode: <10 minutes
   - `vsSML` mode: <60 minutes
   - default mode: <60 minutes

2. `~/vsGibbon/generate_compile_times.py`

   - default mode: <10 minutes

3. `~/vsGibbon/generate_cache_stats.py`

- default mode: <30 minutes

4. `~/vsSML/generate_sml_numbers.py`

   - default mode: ~100 minutes

5. `~/vsGHC/generate_ghc_numbers.py`

   - `small` mode: ~15 minutes
   - default mode: ~100 minutes

8.4.3 Output files

8.4.3.1 CSV files

1. `~/vsGibbon/generate_runtimes.py`

   - `vsSML` mode: .csv files are written to `~/vsGibbon/large_sml` (for comparison to MLton)
   - `small` mode: .csv files are written to `~/vsGibbon/small` (for comparison to GHC)
   - default mode: .csv files are written to `~/vsGibbon/large` (for comparison to GHC)

2. `~/vsGibbon/generate_cache_stats.py`

   - default mode: .csv files are written to `~/vsGibbon/large`. It outputs three .csv files for Table 5.

8.4.3.2 PDF files

1. `~/vsGibbon/generate_compile_times.py` outputs three PDF files:

   - `FilterBlogCompileTimes.pdf`,
   - `ContentSearchCompileTimes.pdf`, and
   - `TagSearchCompileTimes.pdf` for the three compile time subfigures in Figure 9 respectively. The files will be written in the working directory.

2. `~/vsSML/generate_sml_numbers.py` outputs three PDF files:

   - `SpeedupMarmosetSmlFilterBlogs.pdf`,
   - `SpeedupMarmosetSmlContentSearch.pdf`, and
   - `SpeedupMarmosetSmlTagSearch.pdf`. The files will be written in `~/vsSML/large`.

3. `~/vsGHC/generate_ghc_numbers.py` outputs three PDF files:

   - `SpeedupMarmosetGhcFilterBlogs.pdf`,
   - `SpeedupMarmosetGhcContentSearch.pdf`, and
   - `SpeedupMarmosetGhcTagSearch.pdf`.
     The files will be written in `~/vsGHC/large` or `~/vsGHC/small` depending on which mode we run in.

8.4.3.3 Mapping of CSV files to the tables in the paper

For every table, we show below how the row and column names in the paper map on the CSV files and on the filenames of the benchmarks in the artifact. The filenames are either for source files (if compiled with Gibbon) or for binaries produced by Marmoset from one of the versions of the source files (the particular version can be decoded from the binary name, e.g.: `layout2ListLenGreedy` is compiled from `layout2ListLen.hs`).

Generally, the names of CSV files do not much the table numbers in the paper (e.g. the `ListLength` numbers are reported in the `Table2.csv` file, while in the paper these numbers are a part of Table 1), but the mapping below resolves the mismatches. Another difference is that our scripts print the median times along with the mean and lower and upper bounds of the confidence interval that are reported in the paper.

- Table 1
  - Line `ListLength` maps to `Table2.csv`
    * `List` maps to `layout2ListLen.hs`
    * `List'` maps to `layout1ListLen.hs`
    * `M_Greedy` maps to `layout2ListLenGreedy`
    * `M_Solver` maps to `layout2ListLenSolver`
  - Line `LogicEval` maps to `Table3.csv`
    * `lr` maps to `eval_l.hs`
    * `rl` maps to `eval_r.hs`
    * `M_Greedy` maps to `eval_rGreedy`
    * `M_Solver` maps to `eval_rSolver`
  - Line `Rightmost`: `Table5.csv`
    * `lr` maps to `TreeRightMost_l.hs`
    * `rl` maps to `TreeRightMost_r.hs`
    * `M_Greedy` maps to `TreeRightMost_lGreedy`
    * `M_Solver` maps to `TreeRightMost_lSolver`
- Table 2
  - Line `AddOneTree`: `Table4a.csv`
    * `Misalgn_pre` maps to `TreeAddOnePrePost.hs`
    * `Algn_pre` maps to `TreeAddOnePre.hs`
    * `Algn_in` maps to `TreeAddOneIn.hs`
    * `Algn_post` maps to `TreeAddOnePost.hs`
    * `M_Greedy` maps to `TreeAddOnePreGreedy`
    * `M_Solver` maps to `TreeAddOnePreSolver`
  - Line `ExpTree`: `Table4b.csv`
    * `Misalgn_pre` maps to `TreeExpoPrePost.hs`
    * `Algn_pre` maps to `TreeExpoPre.hs`
    * `Algn_in` maps to `TreeExpoIn.hs`
    * `Algn_post` maps to `TreeExpoPost.hs`
    * `M_Greedy` maps to `TreeExpoPreGreedy`
    * `M_Solver` maps to `TreeExpoPreSolver`

- Line CopyTree: Table4c.csv
  - Misalgn_pre maps to TreeCopyPrePost.hs
  - Algn_pre maps to TreeCopyPre.hs
  - Algn_in maps to TreeCopyIn.hs
  - Algn_post maps to TreeCopyPost.hs
  - M_Greedy maps to TreeCopyPreGreedy
  - M_Solver maps to TreeCopyPreSolver

- Table 3
  - Line FilterBlogs: Table6a.csv
    - hiadctb maps to layout1FilterBlogs.hs
    - ctbhiad maps to layout2FilterBlogs.hs
    - tbchiad maps to layout3FilterBlogs.hs
    - tcbhiad maps to layout4FilterBlogs.hs
    - btchiad maps to layout5FilterBlogs.hs
    - bchiadt maps to layout7FilterBlogs.hs
    - cbiadht maps to layout8FilterBlogs.hs
    - M_Greedy maps to layout8FilterBlogsGreedy
    - M_Solver maps to layout8FilterBlogsSolver
  - Line EmphContent: Table6b.csv
    - hiadctb maps to layout1ContentSearch.hs
    - ctbhiad maps to layout2ContentSearch.hs
    - tbchiad maps to layout3ContentSearch.hs
    - tcbhiad maps to layout4ContentSearch.hs
    - btchiad maps to layout5ContentSearch.hs
    - bchiadt maps to layout7ContentSearch.hs
    - cbiadht maps to layout8ContentSearch.hs
    - M_Greedy maps to layout8ContentSearchGreedy
    - M_Solver maps to layout8ContentSearchSolver
  - Line TagSearch: Table6c.csv
    - hiadctb maps to layout1TagSearch.hs
    - ctbhiad maps to layout2TagSearch.hs
    - tbchiad maps to layout3TagSearch.hs
    - tcbhiad maps to layout4TagSearch.hs
    - btchiad maps to layout5TagSearch.hs
    - bchiadt maps to layout7TagSearch.hs
    - cbiadht maps to layout8TagSearch.hs
    - M_Greedy maps to layout8TagSearchGreedy
    - M_Solver maps to layout8TagSearchSolver

- Table 4
  - Line FilterBlogs: Table7a.csv
    - Gibbon maps to manyFuncs-FilterBlogs
    - M_Greedy maps to manyFuncsGreedy-FilterBlogs
    - M_Solver maps to manyFuncsSolver-FilterBlogs

  - Line `EmphContent`: `Table7b.csv`
    * `Gibbon` maps to `manyFuncs-EmphKeyword`
    * `M_Greedy` maps to `manyFuncsGreedy-EmphKeyword`
    * `M_Solver` maps to `manyFuncsSolver-EmphKeyword`
  - Line `TagSearch`: `Table7c.csv`
    * `Gibbon` maps to `manyFuncs-EmphKeywordInTag`
    * `M_Greedy` maps to `manyFuncsGreedy-EmphKeywordInTag`
    * `M_Solver` maps to `manyFuncsSolver-EmphKeywordInTag`

- Table 5
  - Line `FilterBlogs`: `Table8a.csv` (if run outside of the docker using the cache script)
    * `hiadctb` maps to `layout1FilterBlogs.hs`
    * `ctbhiad` maps to `layout2FilterBlogs.hs`
    * `tbchiad` maps to `layout3FilterBlogs.hs`
    * `tcbhiad` maps to `layout4FilterBlogs.hs`
    * `btchiad` maps to `layout5FilterBlogs.hs`
    * `bchiadt` maps to `layout7FilterBlogs.hs`
    * `cbiadht` maps to `layout8FilterBlogs.hs`
    * `M_Greedy` maps to `layout8FilterBlogsGreedy`
    * `M_Solver` maps to `layout8FilterBlogsSolver`
  - Line `EmphContent`: `Table8b.csv`
    * `hiadctb` maps to `layout1ContentSearch.hs`
    * `ctbhiad` maps to `layout2ContentSearch.hs`
    * `tbchiad` maps to `layout3ContentSearch.hs`
    * `tcbhiad` maps to `layout4ContentSearch.hs`
    * `btchiad` maps to `layout5ContentSearch.hs`
    * `bchiadt` maps to `layout7ContentSearch.hs`
    * `cbiadht` maps to `layout8ContentSearch.hs`
    * `M_Greedy` maps to `layout8ContentSearchGreedy`
    * `M_Solver` maps to `layout8ContentSearchSolver`
  - Line `TagSearch`: `Table8c.csv`
    * `hiadctb` maps to `layout1TagSearch.hs`
    * `ctbhiad` maps to `layout2TagSearch.hs`
    * `tbchiad` maps to `layout3TagSearch.hs`
    * `tcbhiad` maps to `layout4TagSearch.hs`
    * `btchiad` maps to `layout5TagSearch.hs`
    * `bchiadt` maps to `layout7TagSearch.hs`
    * `cbiadht` maps to `layout8TagSearch.hs`
    * `M_Greedy` maps to `layout8TagSearchGreedy`
    * `M_Solver` maps to `layout8TagSearchSolver`

8.4.4 Miscellaneous

- The output from `~/vsGibbon/generate_runtimes.py` is written to CSV files and stdout. However, the output written to stdout may be compressed (. . . between columns in the tables means compressed output). The user may try to decrease the font size in the terminal to see the full output. In any case, the CSV files have the full output.

- Some scripts output PDF files, which can be transferred out of the container using `docker cp` command, in order to view them.
- The following scripts can take an additional `--verbose` flag to show extra output while the script is running.
- `~/vsGibbon/generate_runtimes.py`
- `~/vsGHC/generate_sml_numbers.py`
- `~/vsGHC/generate_ghc_numbers.py`

8.4.5 Build Marmoset and PAPI outside Docker for generating Table 5

Install dependencies to build Marmoset on Ubunutu 22.04:

```
$ sudo apt-get update
$ sudo apt-get install software-properties-common \
                       libgc-dev \
                       libgmp-dev \
                       build-essential \
                       uthash-dev \
                       vim wget curl
```

Install Racket:

```
$ wget --no-check-certificate \
      https://mirror.racket-lang.org/installers/7.5/racket-7.5-x86_64-linux.sh
$ chmod +x racket-7.5-x86_64-linux.sh
$ ./racket-7.5-x86_64-linux.sh
```

Install the Haskell toolchain:

```
$ curl --proto '=https' --tlsv1.2 -sSf https://get-ghcup.haskell.org \
   | BOOTSTRAP_HASKELL_NONINTERACTIVE=1 \
   BOOTSTRAP_HASKELL_GHC_VERSION=9.4.6 \
   BOOTSTRAP_HASKELL_CABAL_VERSION=3.8.1.0 \
   BOOTSTRAP_HASKELL_INSTALL_STACK=1 \
   BOOTSTRAP_HASKELL_INSTALL_HLS=1 \
   BOOTSTRAP_HASKELL_ADJUST_BASHRC=P sh
```

Install the Rust toolchain:

```
$ curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs \
      | sh -s -- -y --default-toolchain=1.71.0
```

Build Marmoset:

```
$ tar -xf marmoset.tar
$ cd marmoset && source set_env.sh
$ cd gibbon-compiler && cabal v2-build exe:gibbon && cabal v2-install exe:gibbon
```

Install PAPI:

```
$ wget https://github.com/icl-utk-edu/papi/archive/refs/tags/papi-7-1-0-t.tar.gz && \
    mkdir papi && \
    tar -xvzf papi-7-1-0-t.tar.gz -C papi && \
    cd papi && cd papi-papi-7-1-0-t && cd src && \
    ./configure && make -j10 && make install
$ export PAPI_EVENTS="PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L2_DCM"
```

- Run `vsGibbon/generate_cache_stats.py` to generate Table 8 of the paper.

8.5 Proof of a reusable badge

The implementation of Marmoset is currently under review for inclusion in the main line of the Gibbon project, which is open source. The way to build our code is shown in the Dockerfile, and it is identical to the standard instructions to build Gibbon. The benchmarks are available as the part of this artifact.

8.5.1 Structure of the Marmoset implementation

Marmoset is an extension to Gibbon, an open source compiler written in Haskell. Gibbon compiles high level programs written in a subset of Haskell to operate on serialized data in memory. The compiler is written as a series of micro passes that do a small amount of work. Marmoset is implemented as a combination of passes in that pipeline. It is straightforward to register and write a pass in the compiler. Hence our framework can be extended with more complex optimizations without substantial changes to the compiler. This makes future research easy to build on top of the current framework.

The solver used in Marmoset is open source, and the solver can be changed for other solvers.

Marmoset consists of the following modules extending Gibbon (files locations relative to `~/marmoset/gibbon-compiler/src` in the container):

- `Gibbon/Passes/ControlFlowGraph.hs` – This pass adds a static analysis to generate the control flow graph of the functions in the program.
- `Gibbon/Passes/DefinitionUseChains.hs` – This pass does a def-use, use-def chains analysis for each function.
- `Gibbon/Passes/CallGraph.hs` – This pass generates the call graph from the program.
- `Gibbon/Passes/AccessPatternsAnalysis.hs` – This pass generates a graph recording the access patterns between fields of a data constructor for each function in the program.
- `Gibbon/Passes/SolveLayoutConstrs.hs` – This pass generates the ILP constraints and calls the solver.
- `Gibbon/Passes/OptimizeADTLayout.hs` – This pass optimizes the layout of each data constructor globally.

References

1 Creative commons attribution 4.0 international license. `https://creativecommons.org/licenses/by/4.0/`. Accessed: 2024-07-12.

2 Shirley Browne, Jack Dongarra, Nathan Garner, George Ho, and Phil Mucci. A portable programming interface for performance evaluation on modern processors. *The International Journal of High Performance Computing Applications*, 14(3):189–204, 2000-09 2000. `doi:10.1177/109434200001400303`.

3 Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

4 Vidush Singhal, Chaitanya Koparkar, Joseph Zullo, Artem Pelenitsyn, Michael Vollmer, Mike Rainey, Ryan Newton, and Milind Kulkarni. Optimizing layout of recursive datatypes with marmoset, 2024. `arXiv:2405.17590`.

5 Michael Vollmer, Chaitanya Koparkar, Mike Rainey, Laith Sakka, Milind Kulkarni, and Ryan R. Newton. Local: a language for programs operating on serialized data. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 48–62, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3314221.3314631`.

6 Michael Vollmer, Sarah Spall, Buddhika Chamith, Laith Sakka, Chaitanya Koparkar, Milind Kulkarni, Sam Tobin-Hochstadt, and Ryan R. Newton. Compiling Tree Transforms to Operate on Packed Representations. In Peter Müller, editor, *31st European Conference on Object-Oriented Programming (ECOOP 2017)*, volume 74 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:29, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.ECOOP.2017.26`.