Reproducibility Report for ACM SIGMOD 2021 Paper: "One `WITH RECURSIVE` is Worth Many `GOTO`s"

DMYTRO BOGATOV, Boston University, United States

This reproducibility work is excellent. All scripts and even paper sources are packaged in a Docker image with an interactive command-line entry-point. Three heat maps and a table were reproduced with values close to the original, and all relationships were maintained.

1 INTRODUCTION

This is a reproducibility report for a paper by Hirn and Grust [2]. To summarize, three heat maps and a table ([2, Figure 18 and Table 1]) have been reproduced accurately enough. The reproducibility scripts are easy to use, intuitive and interactive.

2 SUBMISSION

Reproducibility submission consists of detailed instructions on how to rerun the experiments and a `Makefile` that acts as a command-line entry point for the reviewer. `Makefile` main routine spins up a Docker composition. Although the composition consists of only one service, with this approach, the parameters of the Docker container are declared in a structured file rather than buried in a script as CLI options. The script interactively walks the reviewer through the artifacts that can be generated with options for quick and full execution. A single experiment results in a PDF of a plot or a CSV file with rows of a table, and a special source file to be used by LaTeX to generate the PDF of the paper.

The submission contains:

- Docker image: `kryonix/sigmod2021-experiments:latest`
  - Digest at the time of reproducibility review: `sha256:8bde245...bbb92830`
  - The image contains the scripts to run the experiments and the LaTeX code of the paper
- A `Makefile` is included in the submission, but not available publicly
- A PDF with detailed instructions is included in the submission, but not available publicly
- GitHub repository is referenced from the original paper
  - `github.com/One-WITH-RECURSIVE-is-Worth-Many-GOTOs`
  - the compiler itself does not seem to be a part of the code (nor the reproducibility submission)

3 HARDWARE AND SOFTWARE ENVIRONMENT

Table 1 describes the resources the original paper and the reproducibility reviewer used to run the experiments.

Table 1. Hardware & Software environment

	Original aper [2]	This reproducibility report
Platform	64-bit Linux x86	macOS 12.0.1 Docker 20.10.11
Machine	Unknown	iMac Pro (2017)
CPU model	Intel Core i7	Intel Xeon W
CPU cores	8 physical	8 physical (Docker: 12 virtual)
CPU frequency	3.66 GHz	3.2 GHz
RAM	64 GB	32 GB (Docker: 22 GB)

4 REPRODUCIBILITY EVALUATION

4.1 Process

All experiments have been run using the same method — after executing `make`, interactively choosing a particular artifact. Only the full versions of the experiments were run, I assume the quicker versions simply do a subset of the full version's work. It was possible to follow the reproducibility instructions without the authors' help.

4.2 Results

See reproduced artifacts in Appendix A.

Figures. All three subfigures of [2, Figure 18] have been reproduced. The obtained numbers and the visual heat maps appear to be close enough to the reported values in the paper [2]. The deviation is attributed to the differences in hardware and platform. See Fig. 1.

I have also run a single "quick" figure `global` and a "full" figure `items`. I have not fully examined all the figures except `force`, `bbox` and `vm`, since there are no reference heat maps in the original paper to compare against. Nevertheless, the ability to fully or quickly generate these extra figures is appreciated.

Table. The reproduced absolute values for [2, Table 1] experiments expectedly differ from the ones reported in the paper, but the relations are consistent. Most of all, the speedup (the "Runtime (speedup) after compilation" in [2, Table 1]) is close to the reported value. See Table 2.

Interactive prompt. The few examples from the reproducibility submission have been successfully run in the supplied interactive prompt. After changing a few numbers in the snippet, the code still executes correctly.

5 SUMMARY

All figures and a table have been reproduced. The ideas, claims, and findings supported by these figures are therefore reproduced as well.

The work on reproducibility is exemplary: the code is trivial to run, the interactive console UI guides the user, the figures are generated as vector PDFs, and the scripts even let one compile a full paper from the results of the experiments.[1]

REFERENCES

[1] Ravindra Guravannavar and S. Sudarshan. 2008. Rewriting Procedures for Batched Bindings. *Proc. VLDB Endow.* 1, 1 (aug 2008), 1107–1123. https://doi.org/10.14778/1453856.1453975

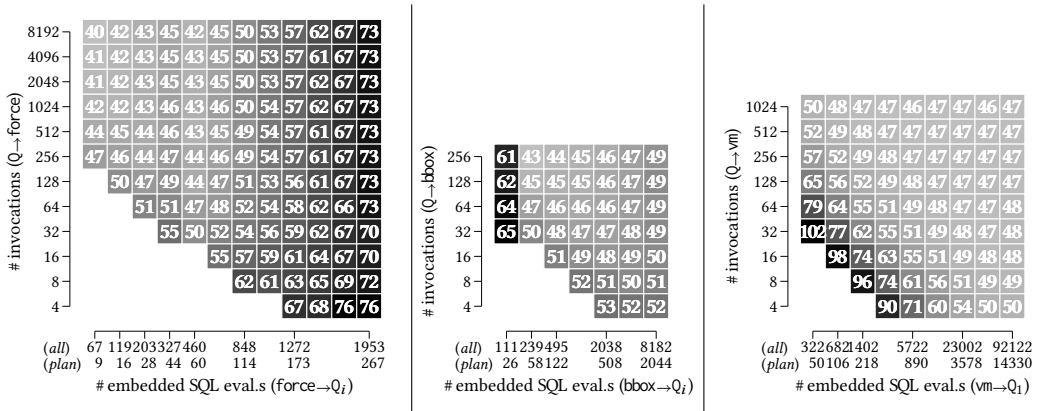
[2] Denis Hirn and Torsten Grust. 2021. *One WITH RECURSIVE is Worth Many GOTOs.* Association for Computing Machinery, New York, NY, USA, 723–735. https://doi.org/10.1145/3448016.3457272

[3] Holtsetio. 2019. MySQL Raytracer. https://demozoo.org/productions/268459/.

[4] Varun Simhadri, Karthik Ramachandra, Arun Chaitanya, Ravindra Guravannavar, and S. Sudarshan. 2014. Decorrelation of user defined function invocations in queries. In *2014 IEEE 30th International Conference on Data Engineering*. 532–543. https://doi.org/10.1109/ICDE.2014.6816679

[1] Thanks for including the LaTeX code in the image. With it, I was able to generate your figures and the table in this report with the original formatting.

A REPRODUCED ARTIFACTS



(a) Heat map for UDF force. (b) Heat map for UDF bbox. (c) Heat map for UDF vm.

Fig. 1. **Reproduced [2, Figure 18].** Runtime (in % of PL/SQL UDFs) after compilation, Q → f and f → Q_i context switches varied (lower/lighter is better).

Table 2. **Reproduced [2, Table 1].** A collection of PL/SQL UDFs with context switching overhead before and speedup after compilation to SQL.

UDF		Return type	$\lvert Q_i \rvert$	LOC	Loop constructs	CC	Q → f+f → Q_i overhead	Runtime (speedup) after compilation		Trampoline transitions
bbox	detect bounding box of a 2D object	box	2	41	1 ␣	5	34.7%	47.09%	(2.12×)	1157
force	*n*-body simulation (Barnes-Hut quad tree)	point	3	43	1 ␣	5	50.1%	50.4%	(1.98×)	263
global	does TPC-H order ship intercontinentally?	boolean	1	20	1 ␣	3	35.0%	64.6%	(1.55×)	9
items	count items in hierarchy (adapted from [1])	int	2	27	1 ␣	2	55.8%	30.53%	(3.28×)	4097
late	find delayed orders (transcribed TPC-H Q21)	boolean	1	25	1 ␣	4	31.9%	74.46%	(1.34×)	9
march	track border of 2D object (Marching Squares)	point[]	2	40	1 ␣	5	32.1%	67.76%	(1.48×)	4568
margin	buy/sell TPC-H orders to maximize margin	row	3	57	1 ␣	5	18.2%	85.19%	(1.17×)	59
markov	Markov-chain based robot control	int	3	44	1 ␣	3	17.7%	71.91%	(1.39×)	1026
packing	pack TPC-H lineitems tightly into containers	int[][]	3	82	3 ␣␣	9	45.8%	60.02%	(1.67×)	312
savings	optimize supply chain of a TPC-H order	row	6	66	1 ␣	4	40.2%	39.58%	(2.53×)	9
sched	schedule production of TPC-H lineitems	row array	5	77	2 ␣	6	31.4%	65.08%	(1.54×)	33
service	determine service level (taken from [4])	text	1	22	0 ∅	3	27.2%	50.58%	(1.98×)	0
ship	customer's preferred shipping mode	text	3	34	0 ∅	3	17.4%	72.23%	(1.38×)	0
sight	compute polygon seen by point light source	polygon	3	49	2 ␣	3	69.5%	88.77%	(1.13×)	662
visible	derive visibility in a 3D hilly landscape	boolean	2	57	1 ␣	3	46.9%	55.23%	(1.81×)	258
vm	execute program on a simple virtual machine	numeric	1	28	1 ␣	17	39.1%	48.84%	(2.05×)	7166
ray	complete PL/SQL ray tracer (adapted from [3])	int[]	5	230	4 ␣␣	25	22.9%	357.9%	(0.28×)	59642
sheet	evaluate inter-dependent spreadsheet formulae	float	9	117	4 ␣␣␣	19	27.7%	173.19%	(0.58×)	2811