Hidden assumptions in static verification of data-race free GPU programs

Tiago Cogumbreiro[1][0000−0002−3209−9258] and Julien Lange[2][0000−0001−9697−1378]

[1] University of Massachusetts Boston, USA
tiago.cogumbreiro@umb.edu
[2] Royal Holloway, University of London, UK
julien.lange@rhul.ac.uk

Abstract. GPUs are massively parallel devices that promise a great return of investment at a cost: GPUs are notably difficult to get right. We discuss a static analysis tool for GPU programs, called Faial, that can detect data-races and data-race freedom. We studied a dataset of 191 data-race free programs and found that 98% needs specific thread configuration to be analyzable, and that 27% needs user-provided assertions to be analyzable. We also report that Faial was able to find data-races in at least 92% of the kernels with missing assumptions.

1 Introduction

For the last 20 years, Vivek Sarkar has been studying the problem of analyzing a data-races in parallel programs both statically [6, 28–31] and dynamically [8, 12, 17, 26, 27, 32]. A data-race is a bug characterized by two unsynchronized memory accesses targeting the same location by different threads, where at least one access is a store. This paper focuses on data-races that arise in the context of GPU programs (also called kernels). GPUs have been widely successful in propelling the scientific advancement of a series of research fields, such as Artificial Intelligence, Machine Learning, molecular modeling, systems biology, and medical imaging.

Data-race detection is a program verification technique that proves the existence of a data-race in a possible run of a program. The most common approach to detect data-races is with dynamic analysis, by monitoring the execution of the program to find data-races. Many dynamic analysis techniques have been proposed [13, 16, 18, 23, 24, 34, 35]. However, since the runtime overhead of dynamic analysis is of 10× up to 1,000× and require the program's input, dynamic analysis is more applicable to testing. Symbolic execution and model checking can be used to detect data-races without needing the program's input, however the overheads can be even higher due to the state explosion problem [21, 22, 25].

Data-races can also be detected statically, thus sidestepping the runtime overheads. *Data-race freedom (DRF) detectors* for GPU programs [4, 5, 9, 10, 19, 20] can guarantee that a program is free from data-races, in the analysis of GPU programs. When a DRF detector is unable to prove that a program is

```
__global__ void saxpy(int n, float a, float *x, float *y) {
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}
```

Fig. 1: A simple GPU program.

DRF, it generates an alarm that represents a *potential* data-race, *i.e.*, alarms may be spurious. Such techniques can be used to find data-races, by manually validating the alarms. Yet, since these tools are *unable* to guarantee that the alarms are true, we do not consider this family of tools to be data-race detectors. To the best of our knowledge, and excluding symbolic execution and model checking approaches, Yuki *et al.* were among the first to introduce a static race detector [33], for X10 programs. Chatarasi *et al.* proposed the first static race detector for OpenMP and openACC [7], Gorogiannis *et al.* introduce the first static race detector for multithreaded programs [14], and Liew *et al.* introduce the first static race detector for GPU programs [**?**].

In this paper we evaluate Faial [**?**], a data-race and DRF detector for GPU programs. We investigate how different features of the analysis affect DRF detection in a dataset that only contains data-race free kernels. We also investigate whether the tool can report data-races when it lacks information to prove DRF.

The outline of this paper is as follows. Section 2 gives some background by introducing GPU programming as well as discussing implicit assumptions that are needed to prove DRF. Section 3 introduces and tests our research questions. Finally, Section 4 summarizes our findings.

2 Background

In this section, we give a quick background on GPU programming. We then show that even trivial GPU programs include multiple implicit assumptions.

2.1 GPU programming

SAXPY (Single-Precision $A \cdot X$ Plus Y) is a classic example that showcases the kind of numeric applications that run on GPU devices. Given two vectors of floating points X and Y and a scalar A the program updates vector Y such that $Y[i]$ stores the result of $A \cdot X[i] + Y[i]$ for each element i. A SAXPY operation can be implemented as a GPU program in Figure 1. In this paper we use the CUDA Application Programming Interface (API); the same concepts apply to other GPU programming models. A GPU executes function `saxpy` for a certain number of threads arranged in groups. Each group of threads is called a *block*. The threads of a block are logically arranged in a 3-D space, each thread is uniquely identified by a 3D point accessible with variable `threadIdx`. The set of

all blocks is also logically arranged in a 3-D space, each block is uniquely identified with a 3-D point accessible with variable `blockIdx`. The number of threads per block, *i.e.*, the block layout, is accessible in variable `blockDim`. The number of blocks in the system are given by variable `gridDim`. A GPU program runs a copy of function `saxpy` per thread in parallel, instantiating variables `threadIdx` and `blockIdx` for each thread. Variable i represents a unique thread across all groups, since it projects the x-component of variables `blockIdx` with `threadIdx` onto a linear space. A *thread configuration* is defined as the number of blocks and the number of threads per block.

When a kernel is data-race free only under certain assumptions we call that kernel *partially data-race free*. For instance, the example in Figure 1, taken from a tutorial on CUDA programming [15], is partially data-race free. Next, we show two assumptions that render Figure 1 partially data-race free: thread configurations, and grid-level synchronization.

Ranging over all thread configurations. The statement that variable i is unique thread across all groups only holds when there is only one dimension in the y and z axis. Hence, a data-race exists between thread $\texttt{threadIdx} = \{\texttt{x} = 0, \texttt{y} = 1, \texttt{z} = 1\}$ and $\texttt{threadIdx} = \{\texttt{x} = 0, \texttt{y} = 0, \texttt{z} = 0\}$ both from block $\texttt{blockIdx} = \{\texttt{x} = 0, \texttt{y} = 0, \texttt{z} = 0\}$ for a block $\texttt{blockDim} = \{\texttt{x} = 1, \texttt{y} = 2, \texttt{z} = 2\}$, *i.e.*, 2×2 threads in the y-z axis. The data-race occurs because the projection in variable `i` assumes that all threads are arranged in dimension x, yet a data-race can occur if there are threads in dimensions y and z. We can add an assertion to make this fact explicit:

```
__assume(blockDim.y == 1 && blockDim.z == 1);
```

Grid-level synchronization. The distinction between block-level and grid-level analysis is important to the analysis, because different kinds of memory can be shared at different levels, and also synchronization mechanisms are available at different levels. If we consider data-races across different blocks, then another data-race is possible. For instance, between thread $\texttt{threadIdx} = \{\texttt{x} = 0, \texttt{y} = 0, \texttt{z} = 0\}$ of block $\texttt{blockIdx} = \{\texttt{x} = 0, \texttt{y} = 0, \texttt{z} = 0\}$ and thread $\texttt{threadIdx} = \{\texttt{x} = 0, \texttt{y} = 0, \texttt{z} = 0\}$ of block $\texttt{blockIdx} = \{\texttt{x} = 0, \texttt{y} = 1, \texttt{z} = 1\}$. We can add an assertion to make the data-race freedom assumption explicit: `__assume(gridDim.y == 1 && gridDim.z == 1);`

We list the kernel with both user-provided assertions that are needed to prove data-race freedom in Figure 2.

3 Evaluation

Faial is the only tool capable of data-race and data-race-freedom detection. Given a data-set of kernels identified as data-race free, we pose two research questions:

RQ1: *Which analysis features affect partial data-race freedom?* We select different features and measure how many kernels cannot be analyzed to understand the impact each feature has in this dataset.

```
__global__ void saxpy(int n, float a, float *x, float *y) {
  __assume(blockDim.y == 1 && blockDim.z == 1);
  __assume(gridDim.y == 1 && gridDim.z == 1);
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}
```

Fig. 2: A simple example with user-provided assumptions.

RQ2: *Can static data-race detection help with missing assumptions?* In the context of this experiment, racy kernel indicate missing assumptions. Since Faial is not guaranteed to find every possible data-race, we want to test if we can use Faial to detect data-races in kernels with missing assumptions.

Both research questions consider 5 experiments. Each experiment runs Faial on the same 191 kernels with different analysis settings. The tool can report that the kernel is data-race free, racy, or timeout.

Data selection. The dataset we use is taken from a benchmark suite of GPU kernels [2]. The dataset is well studied as it has been used in multiple published papers on static analysis of data-races in GPU kernels [2, 9**?** , 10]. The dataset consists of CUDA kernels from 4 benchmark suites: NVIDIA GPU Computing SDK v2.0 (8 kernels), NVIDIA GPU Computing SDK v5.0 (165 kernels), Microsoft C++ AMP Sample Projects (20 kernels), gpgpu-sim benchmarks [1] (33 kernels). Every kernel is annotated with verification-specific conditions: a thread configuration and optionally user-provided assumptions.

We pick 191 kernels that are deemed data-race free by Faial. Some kernels include user-provided assertions created by the authors of the dataset [2]. Most commonly, the user-provided assertions are stating that a certain variable has fixed value, for instance that the height of a matrix is of some arbitrary size, say 512. Importantly, the user-provided assertions are not constraining the thread configurations, *e.g.*, like we did in Figure 2.

3.1 RQ1: Which analysis features affect partial data-race freedom?

Table 1 lists the 5 experiments that were performed according to the output of the analysis, data-race free, racy, or timeout.

Discussion. Experiment 1 is our baseline, since all kernels can be checked as data-race free, yet note that grid-level analysis is not performed. In experiment 2, we enable grid-level analysis and note that Faial is unable to analyze 5 kernels. Faial delegates a step of data-race freedom analysis (index equality) to the Z3 [11] Satisfiability Modulo Theories (SMT) solver. We are able to verify *all* kernels by setting the SMT solver's theory to `AUFLIA`, which assumes closed formulas

Table 1: Column Id holds an identifier of the experiment. Column Block states whether block-level synchronization is checked. Column Grid states whether grid-level synchronization is checked. Column Assert states whether user-provided assertions are used. Column FixThr states whether a fixed thread configuration is used. Column DRF counts the kernels identified as data-race free. Column Racy counts the kernels identified as racy. Column Unk counts the kernels where the analysis is unable to detect data-race freedom nor data-races. Column T/O counts the kernels where the analysis timed out. We include the percentage of kernels over the total number of kernels under analysis.

Id	Block	Grid	Assert	FixThr	DRF	(%)	Racy	(%)	Unk	(%)	T/O	(%)
1	Y	N	Y	Y	191	100%	0	0%	0	0%	0	0%
2	Y	Y	Y	Y	186	97%	0	0%	0	0%	5	3%
3	Y	N	N	Y	139	73%	49	26%	3	2%	0	0%
4	Y	N	Y	N	3	2%	173	91%	15	8%	0	0%
5	Y	Y	N	N	2	1%	173	91%	13	7%	3	2%

of linear integer arithmetic extended with free sort and function symbols. In experiment 3, we disable user-provided assertions. Only 26% of the kernels require user-provided annotations to prove data-race freedom.

In experiment 4, we range over all possible thread configurations, rather than using a specific thread configuration. Almost every kernel under analysis (98%) expects a specific thread configuration. Faial would be able to analyze many more kernels fully automatically if it could extract the thread configuration present in the kernel launching codes. Bardsley *et al.* have explored a dynamic analysis technique that extracts the runtime parameters of kernel launches [3].

In experiment 5, we enable grid-level analysis, disable user-provided assertions, and range over all possible thread configurations. We find that there are only 2 kernels that are *fully* data-race free, regardless of the thread configuration and without requiring any user assertions. In one kernel, the only memory accesses are atomics that do not introduce data-races. Atomics are supported by Faial. In the other kernel, the only write access is a benign data-race ignored by Faial. Benign data-races occur when both threads write the same value. Benign data-races are not considered errors. Faial can flag benign data-races as errors if the user chooses.

3.2 RQ2: Can data-race detection help with missing assumptions?

In this research question we examine kernels that are *not* considered data-race free by Faial, so either racy, unknown, or have a timeout. We assess whether our tool can detect data-races when there are missing assumptions, *e.g.*, absent thread configuration.

Discussion. The results in Table 2 show that the vast majority of kernels (91%) with missing assumptions can be detected by Faial. In our experience, having

Table 2: Column Id holds an identifier of the experiment. Column Block states whether block-level synchronization is checked. Column Grid states whether grid-level synchronization is checked. Column Assert states whether user-provided assertions are used. Column FixThr states whether a fixed thread configuration is used. Column Racy/Non-DRF gives the proportion of number of kernels with data-races detected versus the total number of kernels that are not data-race free. Column % gives the percentage of Racy/Non-DRF.

Id	Block	Grid	Assert	FixThr	Racy/Non-DRF	%
3	Y	N	N	Y	49/52	94%
4	Y	N	Y	N	173/188	92%
5	Y	Y	N	N	173/189	92%

a static data-race detector has been quite effective to figuring out the correct analysis settings. In contrast, when relying on the alarms of a data-race-freedom detector, there is always uncertainty whether there is an actual data-race or a spurious one.

3.3 Bugs found

In the course of writing this paper, we discovered bugs in the dataset and in Faial. We added assumptions and changed the thread configurations of 4 kernels, since these triggered data-races when grid-level analysis was enabled. In 3 kernels we had to reduce the level of parallelism, by decreasing the number of thread blocks. In 1 kernels, we added a user-provided assumption, a constraint of a template parameter that was mentioned as a source comment, yet absent. We excluded 6 kernels from our evaluation that were being considered fully data-race free by Faial, although they are not. Two C++ features are currently unsupported by Faial: array addresses being incremented in a loop[3] (affected 3 kernels), and references as function parameters[4] (affected 3 kernels). Since it is quite rare for a kernel to be fully data-race free, experiment 5 proved as an effective sanity check to exercise the correctness of Faial.

4 Conclusion

In this paper we measured the effect of multiple analysis features when detecting data-race freedom statically, in a dataset of 191 data-race free kernels. We found that 98% of the kernels needed a specific thread configuration to be analyzable and that only 27% of the kernels needed user-provided assertions. These results suggest that to enable a fully automatic static analysis, these tools need to be able to infer valid thread configurations. We also showed that the static race detection of Faial was able to find data-races in at least 92% of the kernels studied.

[3] https://gitlab.com/umb-svl/faial/-/issues/117
[4] https://gitlab.com/umb-svl/faial/-/issues/113

The static race detector also helped us identify incorrect thread configurations and missing user-provided assumptions in 4 kernels. Finally, we identified two areas of improvement for Faial: 6 kernels were excluded from the evaluation due to limitations of the tool (arrays being updated in loops and references as function parameters), and setting Faial's default SMT theory to `AUFLIA` fixed 5 timeouts.

Acknowledgments. This material is based upon work supported by the National Science Foundation under Grant No. 2204986. We thank Francis Alcos, Gregory Blike, Ayden Diel, Samyak Gangwal, Austin Guiney, Ramsey Harrison, Paul Maynard, Udaya Sathiyamoorth, and Hannah Zicarelli for their contributions to Faial.

Bibliography

[1] Bakhoda, A., Yuan, G.L., Fung, W.W.L., Wong, H., Aamodt, T.M.: Analyzing CUDA workloads using a detailed GPU simulator. In: Proceedings of ISPASS. pp. 163–174. IEEE, Piscataway, NJ, USA (2009). https://doi.org/10.1109/ISPASS.2009.4919648

[2] Bardsley, E., Betts, A., Chong, N., Collingbourne, P., Deligiannis, P., Donaldson, A.F., Ketema, J., Liew, D., Qadeer, S.: Engineering a static verification tool for GPU kernels. In: Proceedings of CAV. vol. 8559, pp. 226–242. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-319-08867-9_15

[3] Bardsley, E., Donaldson, A.F., Wickerson, J.: KernelInterceptor: Automating GPU kernel verification by intercepting kernels and their parameters. In: Proceedings of IWOCL. pp. 1–5. ACM, New York, NY, USA (5 2014). https://doi.org/10.1145/2664666.2664673

[4] Betts, A., Chong, N., Donaldson, A.F., Ketema, J., Qadeer, S., Thomson, P., Wickerson, J.: The design and implementation of a verification technique for GPU kernels. Transactions on Programming Languages and Systems **37**(3), 1–49 (2015). https://doi.org/10.1145/2743017

[5] Betts, A., Chong, N., Donaldson, A.F., Qadeer, S., Thomson, P.: GPUVerify: a verifier for GPU kernels. In: Proceedings of OOPSLA. pp. 113–132. ACM, New York, NY, USA (2012). https://doi.org/10.1145/2384616.2384625

[6] Chatarasi, P., Shirako, J., Kong, M., Sarkar, V.: An extended polyhedral model for SPMD programs and its use in static data race detection. In: Proceedings of LCPC. LNCS, vol. 10136, pp. 106–120. Springer (2016). https://doi.org/10.1007/978-3-319-52709-3_10

[7] Chatarasi, P., Shirako, J., Kong, M., Sarkar, V.: An extended polyhedral model for SPMD programs and its use in static data race detection. In: Proceedings of LCPC'16. pp. 106–120. Springer, Berlin, Heidelberg (2017). https://doi.org/10.1007/978-3-319-52709-3_10

[8] Choi, J., Lee, K., Loginov, A., O'Callahan, R., Sarkar, V., Sridharan, M.: Efficient and precise datarace detection for multithreaded object-oriented programs. In: Proceedings of PLDI. pp. 258–269. ACM (2002). https://doi.org/10.1145/512529.512560

[9] Cogumbreiro, T., Lange, J., Liew Zhen Rong, D., Zicarelli, H.: Memory access protocols: Certified data-race freedom for GPU kernels. FMSD (2023). https://doi.org/10.1007/s10703-023-00415-0

[10] Cogumbreiro, T., Lange, J., Rong, D.L.Z., Zicarelli, H.: Checking datarace freedom of GPU kernels, compositionally. In: Proceedings of CAV. LNCS, vol. 12759, pp. 403–426. ACM, New York, NY, USA (2021). https://doi.org/10.1007/978-3-030-81685-8_19

[11] De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proceedings of TACAS. pp. 337–340. Springer, Berlin, Heidelberg (2008)

[12] Dimitrov, D.K., Vechev, M.T., Sarkar, V.: Race detection in two dimensions. ACM Transactions on Parallel Computing **4**(4), 1–22 (2018). https://doi.org/10.1145/3264618

[13] Eizenberg, A., Peng, Y., Pigli, T., Mansky, W., Devietti, J.: BARRACUDA: Binary-level Analysis of Runtime RAces in CUDA programs. In: Proceedings of PLDI. pp. 126–140. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3062341.3062342

[14] Gorogiannis, N., O'Hearn, P.W., Sergey, I.: A true positives theorem for a static race detector. Proceedings of the ACM on Programming Languages **3**(POPL), 1–29 (2019). https://doi.org/10.1145/3290370

[15] Harris, M.: An easy introduction to CUDA C and C++. https://developer.nvidia.com/blog/easy-introduction-cuda-c-and-c/ (2012), accessed: July 9, 2024.

[16] Holey, A., Mekkat, V., Zhai, A.: HAccRG: Hardware-accelerated data race detection in GPUs. In: Proceedings of ICPP. pp. 60–69. IEEE, Piscataway, NJ, USA (2013). https://doi.org/10.1109/ICPP.2013.15

[17] Jin, F., Yu, L., Cogumbreiro, T., Shirako, J., Sarkar, V.: Dynamic Determinacy Race Detection for Task-Parallel Programs with Promises. In: Proceedings of ECOOP. LIPIcs, vol. 263, pp. 1–30. Schloss Dagstuhl, Dagstuhl, Germany (2023). https://doi.org/10.4230/LIPIcs.ECOOP.2023.13

[18] Kamath, A.K., George, A.A., Basu, A.: ScoRD: A scoped race detector for GPUs. In: Proceedings of ISCA. pp. 1036–1049. IEEE, Piscataway, NJ, USA (2020). https://doi.org/10.1109/ISCA45697.2020.00088

[19] Li, G., Gopalakrishnan, G.: Scalable SMT-based verification of GPU kernel functions. In: Proceedings of FSE. pp. 187–196. ACM, New York, NY, USA (2010). https://doi.org/10.1145/1882291.1882320

[20] Li, G., Gopalakrishnan, G.: Parameterized verification of GPU kernel programs. In: Proceedings of IPDPSW. pp. 2450–2459. IEEE, Piscataway, NJ, USA (2012). https://doi.org/10.1109/IPDPSW.2012.302

[21] Li, G., Li, P., Sawaya, G., Gopalakrishnan, G., Ghosh, I., Rajan, S.P.: GKLEE: Concolic verification and test generation for GPUs. In: Proceedings of PPoPP. vol. 47, pp. 215–224. ACM, New York, NY, USA (2012). https://doi.org/10.1145/2370036.2145844

[22] Li, P., Li, G., Gopalakrishnan, G.: Practical symbolic race checking of GPU programs. In: Proceedings of SC. pp. 179–190. IEEE, Piscataway, NJ, USA (2014). https://doi.org/10.1109/SC.2014.20

[23] Li, P., Hu, X., Chen, D., Brock, J., Luo, H., Zhang, E.Z., Ding, C.: LD: Low-overhead GPU race detection without access monitoring. Transactions on Architecture and Code Optimization **14**(1), 1–25 (2017). https://doi.org/10.1145/3046678

[24] Peng, Y., Grover, V., Devietti, J.: CURD: A dynamic CUDA race detector. In: Proceedings of PLDI. pp. 390–403. ACM, New York, NY, USA (2018). https://doi.org/10.1145/3192366.3192368

[25] Pereira, P., Albuquerque, H., Marques, H., Silva, I., Carvalho, C., Cordeiro, L., Santos, V., Ferreira, R.: Verifying CUDA programs using SMT-based context-bounded model checking. In: Proceedings of SAC. pp. 1648–1653.

ACM, New York, NY, USA (2016). https://doi.org/10.1145/2851613.2851830

[26] Raman, R., Zhao, J., Sarkar, V., Vechev, M.T., Yahav, E.: Efficient data race detection for async-finish parallelism. Formal Methods in System Design **41**(3), 321–347 (2012). https://doi.org/10.1007/S10703-012-0143-7

[27] Raman, R., Zhao, J., Sarkar, V., Vechev, M.T., Yahav, E.: Scalable and precise dynamic datarace detection for structured parallelism. In: Proceedings of PLDI. pp. 531–542. ACM (2012). https://doi.org/10.1145/2254064.2254127

[28] Surendran, R., Raman, R., Chaudhuri, S., Mellor-Crummey, J.M., Sarkar, V.: Test-driven repair of data races in structured parallel programs. In: Proceedings of PLDI. pp. 15–25. ACM (2014). https://doi.org/10.1145/2594291.2594335

[29] Westbrook, E.M., Zhao, J., Budimlic, Z., Sarkar, V.: Practical permissions for race-free parallelism. In: Proceedings of ECOOP. LNCS, vol. 7313, pp. 614–639. Springer (2012). https://doi.org/10.1007/978-3-642-31057-7_27

[30] Ye, F., Schordan, M., Liao, C., Lin, P., Karlin, I., Sarkar, V.: Using polyhedral analysis to verify OpenMP applications are data race free. In: Laguna, I., Rubio-González, C. (eds.) Proceedings of CORRECTNESS@SC. pp. 42–50. IEEE (2018). https://doi.org/10.1109/CORRECTNESS.2018.00010, https://doi.org/10.1109/Correctness.2018.00010

[31] Yu, L., Jin, F., Protze, J., Sarkar, V.: Leveraging the dynamic program structure tree to detect data races in OpenMP programs. In: Proceedings of Correctness@SC. pp. 54–62. IEEE (2022). https://doi.org/10.1109/CORRECTNESS56720.2022.00012

[32] Yu, L., Sarkar, V.: GT-Race: Graph traversal based data race detection for asynchronous many-task parallelism. In: Aldinucci, M., Padovani, L., Torquati, M. (eds.) Proceedings of Euro-Par. LNCS, vol. 11014, pp. 59–73. Springer (2018). https://doi.org/10.1007/978-3-319-96983-1_5

[33] Yuki, T., Feautrier, P., Rajopadhye, S., Saraswat, V.: Array dataflow analysis for polyhedral X10 programs. In: Proceedings of PPoPP. pp. 23–34. ACM, New York, NY, USA (2013). https://doi.org/10.1145/2442516.2442520

[34] Zheng, M., Ravi, V.T., Qin, F., Agrawal, G.: GRace: A low-overhead mechanism for detecting data races in GPU programs. In: Proceedings of PPoPP. pp. 135–146. ACM, New York, NY, USA (2011). https://doi.org/10.1145/1941553.1941574

[35] Zheng, M., Ravi, V.T., Qin, F., Agrawal, G.: GMRace: Detecting data races in GPU programs via a low-overhead scheme. Transactions on Parallel and Distributed Systems **25**(1), 104–115 (2014). https://doi.org/10.1109/TPDS.2013.44