Scaling data-race freedom analysis with array projections

SERPL 2023

Paul Maynard Tiago Cogumbreiro

Data-races are a major source of concurrency errors in GPU programming. Data-Race-Freedom (DRF) analysis is a static analysis technique that aims to guarantee that for all possible executions of a program, every location in memory being written by one thread cannot be concurrently accessed by another thread. In GPU programming, the vast majority of data-races are caused by accesses to arrays. The state of the art of DRF analysis relies on Satisfiability Modulo Theories (SMT) solvers to test when indices accessing arrays are equal, which is one of the conditions to trigger a data-race [2, 3, 4].

DRF analysis relying on SMT solvers is unable to handle all problem parameters (*e.g.*, the number of threads). The root of this limitation is due to nonlinear arithmetic expressions appearing in array indices, which cannot be tested for equality by current solvers [1].

The following example shows a nonlinear expression being used to access the array `paths`. Note how the indexing expression `i + S * t` includes a multiplication of program variables `S` and `t`. Variables `i` and `t` evaluate differently for different threads, because variable `threadIdx.x` evaluates to a unique numeric identifier per thread. This means that an SMT-based DRF analysis is not able to fully analyze this kernel for data-races.

```
__global__ void kernel(float* paths, int S, int T) {
  unsigned int step = gridDim.x * blockDim.x;
  for (unsigned int i = threadIdx.x; i < S ; i += step) {
    for (unsigned int t = 0; t < T ; t++) {
      paths[i + S * t] = f(t);
    }
  }
}
```

A possible workaround is to replace a variable in the expression by a constant, say replace variable `S` by 1,024, and prove that the program is DRF for this particular instance of `S`. A further step involves performing the analysis for a fixed range of values. However, such an approach is unsatisfactory as the problem quickly grows when multiple program variables need to be instantiated.

Our approach. Many nonlinear array accesses in CUDA are actually projections of multidimensional indexing into a single dimensional array. This is necessary due to limitations of the C/C++ languages, as multidimensional arrays in C/C++ must have their dimensions known at compile time. However, an SMT solver can handle a multidimensional array access when the indices in each of the dimensions of the acess are linear. *Our approach is to automatically rewrite multidimensional projections as multidimensional accesses in our internal representation.* For example, we can rewrite function `kernel` to use a multidimensional indexing, rather than a projection.

```
// Idealized C-like syntax
__global__ void kernel (float* paths[][], int S, int T) {
  // Determine thread ID
  unsigned int step = gridDim.x * blockDim.x;
  for (unsigned int i = threadIdx.x; i < S ; i += step) {
    for (unsigned int t = 0; t < T ; t++) {
      paths[i][t] = f(t);
    }
  }
}
```

Our ongoing work involves automating the translation from a multidimensional projection into a multidimensional access, by

- identifying when a nonlinear expression is an array projection,
- determining the dimensions and the indices of the multidimensional access, and
- transforming the code.

Additionally, we are interested in recording how often such patterns occur in practice, by applying our analysis to a dataset of 6,500 CUDA kernels downloaded from open source software.

References

[1] Paul Beame and Vincent Liew. Toward verifying nonlinear integer arithmetic. *Journal of the ACM*, 66(3), 2019.

[2] Adam Betts, Nathan Chong, Alastair F. Donaldson, Jeroen Ketema, Shaz Qadeer, Paul Thomson, and John Wickerson. The design and implementation of a verification technique for GPU kernels. *Transactions on Programming Languages and Systems*, 37(3):1–49, 2015.

[3] Tiago Cogumbreiro, Julien Lange, Dennis Liew Zhen Rong, and Hannah Zicarelli. Memory access protocols: Certified data-race freedom for GPU kernels. *Formal Methods in System Design*, 2023.

[4] Guodong Li and Ganesh Gopalakrishnan. Scalable SMT-based verification of GPU kernel functions. In *Proceedings of FSE*, pages 187–196, New York, NY, USA, 2010. ACM.