Using Polyhedral Analysis to Verify OpenMP Applications are Data Race Free

—

2nd International Workshop on Software Correctness for HPC Applications, Nov 12, 2018, TX, USA

Fangke Ye[1], Markus Schordan[2], Pei-Hung Lin[2], Chunhua Liao[2], Ian Karlin[2], Vivek Sarkar[1]

[1] Georgia Institute of Technology, GA, USA
[2] Lawrence Livermore National Laboratory, CA, USA

—

External Audience (Unlimited)
LLNL-PRES-760684

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was supported by the LLNL-LDRD Program under Project No. 17-ERD-023. Lawrence Livermore National Security, LLC.

Overview

- Introduction + Examples
- Tool Overview: Testing, Static Analysis, Verification
- Approach for data race detection in our tool DRACO
  - Static analysis combining different approaches
  - Focus in this paper: identify loops for which polyhedral analysis is applicable
  - Structure analysis such that remaining loops can be dispatched to other analysis techniques
- Source-to-source transformations enabling polyhedral analysis
- Evaluation (DataRaceBench + Proxy App AMG 2013)
- Conclusion

Introduction

Definition: What is a data race?

Data races occur when multiple threads perform simulataneous conflicting data accesses to the same memory location without proper synchronization and at least one is a write access.

Reasons for the Existence of Data Races

- A data race exists when synchronization between threads is missing.
- Additional synchronization slows down the execution of a parallel program.
- Data races can be dependent on the thread schedule and can be difficult to reproduce and to detect.

Example 1 - Proxy App AMG 2013

```
int hypre_ParCSRRelax_Cheby(....) {
...
   int num_rows = hypre_CSRMatrixNumRows(A_diag);...
   double *u_data
    = hypre_VectorData(hypre_ParVectorLocalVector(u));...
   double *orig_u;...
   orig_u = hypre_CTAlloc(double, num_rows);...
   double *ds_data, *tmp_data;...
   ds_data
    = hypre_VectorData(hypre_ParVectorLocalVector(ds));...
...
#ifdef HYPRE_USING_OPENMP
#pragma omp parallel for private(j) schedule(static)
#endif
      for ( j = 0; j < num_rows; j++ )
      {
         u_data[j] = orig_u[j] + ds_data[j]*u_data[j];
      }
...
}
```

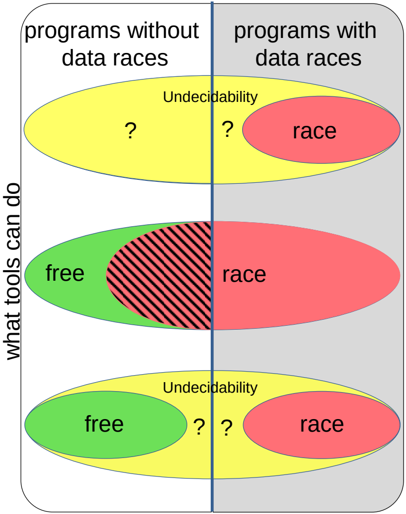
Figure: AMG2013 function `hypre_parCSRRelax_cheby` in file `parcsr_ls/par_relax_more.c`. The directive `omp parallel for` is at line 742 in the original file. This loop is verified to have no data race.

Example 2: DRB043 Polyhedral Loop Nest

```
static void kernel_adi(int tsteps,int n,double X[500 + 0][500 + 0],double A[500 + 0][500 + 0],double B[500 + 0][500 + 0])
{
    int c0;
    int c2;
    int c8;
    for (c0 = 0; c0 <= 9; c0++) {
#pragma omp parallel for private(c8)
      for (c2 = 0; c2 <= 499; c2++) {
        for (c8 = 1; c8 <= 499; c8++) {
          B[c2][c8] = B[c2][c8] - A[c2][c8] * A[c2][c8] / B[c2][c8 - 1];
        }
        for (c8 = 1; c8 <= 499; c8++) {
          X[c2][c8] = X[c2][c8] - X[c2][c8 - 1] * A[c2][c8] / B[c2][c8 - 1];
        }
        for (c8 = 0; c8 <= 497; c8++) {
          X[c2][500 - c8 - 2] = (X[c2][500 - 2 - c8] - X[c2][500 - 2 - c8 - 1] * A[c2][500 - c8 - 3]) / B[c2][500 - 3 - c8];
        }
      }
#pragma omp parallel for
      for (c2 = 0; c2 <= 499; c2++) {
        X[c2][500 - 1] = X[c2][500 - 1] / B[c2][500 - 1];
      }
#pragma omp parallel for private(c8)
      for (c2 = 0; c2 <= 499; c2++) {
        for (c8 = 1; c8 <= 499; c8++) {
          B[c8][c2] = B[c8][c2] - A[c8][c2] * A[c8][c2] / B[c8 - 1][c2];
        }
        for (c8 = 1; c8 <= 499; c8++) {
          X[c8][c2] = X[c8][c2] - X[c8 - 1][c2] * A[c8][c2] / B[c8 - 1][c2];
        }
        for (c8 = 0; c8 <= 497; c8++) {
          X[500 - 2 - c8][c2] = (X[500 - 2 - c8][c2] - X[500 - c8 - 3][c2] * A[500 - 3 - c8][c2]) / B[500 - 2 - c8][c2];
        }
      }
#pragma omp parallel for
      for (c2 = 0; c2 <= 499; c2++) {
        X[500 - 1][c2] = X[500 - 1][c2] / B[500 - 1][c2];
      }
    }
}
```

Data race free!

Testing, Static Analysis, Verification



- Software Testing tools
  - Google Thread Sanitizer (state-of-the-art according to [Effinger-Dean et al., 2012])
  - FastTrack(Java)/Aikido(C) (state-of-the-art according to [Effinger-Dean et al., 2012])
  - Archer - LLNL (based on Google's Thread Sanitizer, supports OpenMP) [Protze, Atzeni, Ahn, et al., 2014]
- Static Software Analyzer
  - LOCKSMITH (subset of C) [Pratikakis et al., 2006]
  - Relay (C code) [Voung et al., 2007]
- Software Verifier
  - BLAST: abstractions (nesC code) [Henzinger, 2003]
  - CHESS: **stateless** bounded MC [Musuvathi, 2008]
  - ompVerify (polyhedral loops) [Basupalli, 2011]
  - CIVL: symbolic execution, no abstraction [Zheng et al., 2015]

Approach in DRACO

Analysis

1. Check for-loop is parallelized by a supported OpenMP directive
2. Check OpenMP pragma contains no unsupported OpenMP directive or clause
3. Pointer analysis: the arrays referenced in the loop nest do not overlap (not implemented yet)
4. Apply analysis-enabling program transformations for polyhedral analysis
5. Polyhedral analysis: if a parallel loop's sequential version can be represented by the polyhedral model
6. Bounds analysis: Check there is no out-of-bounds array access in the loop nest in multi dimensional arrays.

Source-to-source Transformation Enabling Polyhedral Analysis

```
int a[10];

for (i = 0; i < 5; i++)
  for (j = 0; j < 5; j++)
    if (i >= j)
      a[i - j - 1]++;
```

(a) The original loop.

```
int a[10];
for (t = 0; t < 2; t++)
  for (i = 0; i < 5; i++)
    for (j = 0; j < 5; j++)
      if (i >= j)
        if (i - j - 1 < 0 ||
            i - j - 1 >= 10)
          tmp = 0;
```

(b) The transformed loop.

Figure: A transformation example for bounds checking.

The array access is transformed into an if-stmt checking the index-expression's bounds and an outermost loop with two iterations is added. If there is no dependency on a write to 'tmp' between iterations then we can conclude the index-expression is in-bounds.

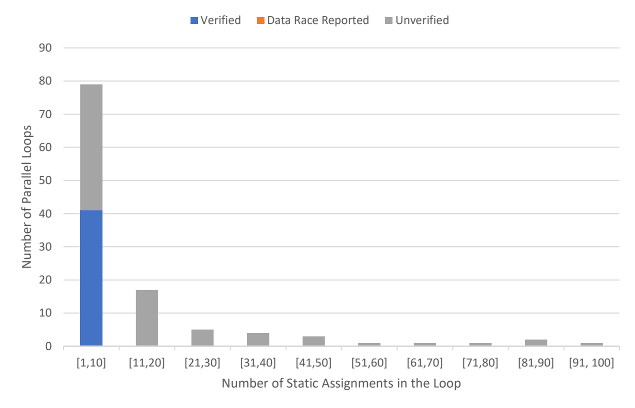
Evaluation: DataRaceBench Results

DataRaceBench 1.2.0 - Verification results

	Data race detected	Verified Data Race Free	Unverified	No Parallel Loop	Total
#benchmarks	26	20	42	28	116
Min. Runtime (s)	0.26	0.24	0.17	0.20	N/A
Max. Runtime (s)	0.63	1.32	208.44	7.17	N/A
Tot. Runtime (s)	12.95	7.83	233.46	25.16	279.40

Evaluation: AMG2013 Results

Analyzed loops (Loop Size Histogram)



Verified: 41 of 114 loops in 0:05:08.4

Conclusion

- DRACO utilizes polyhedral analysis for statically analyzing parallel OpenMP loops.
- DRACO reports for each parallel loop a 3-valued result:
  - verified (no data race exists for any loop bounds or values)
  - data race detected (definitely exists)
  - unverified (due to detected unsupported OpenMP features or over-approximation and potential data race)
- Key to utilizing polyhedral analysis is a precise pointer analysis.
- For non-polyhedral loops model checking techniques will be used for verification.