Static Data Race Detection for SPMD Programs via an Extended Polyhedral Representation

Prasanth Chatarasi, Jun Shirako, Vivek Sarkar

Habanero Extreme Scale Software Research Group
Department of Computer Science
Rice University

6th International Workshop on Polyhedral Compilation Techniques (IMPACT'16)

January 19, 2016

Introduction

- Moving towards homogeneous and heterogeneous many-core processors
  - 100's of cores per chip
  - Performance driven by parallelism
  - Constrained by energy and data movement
- Need for improved productivity and scalability in parallel programming models
- Most successful model - **Single Program Multiple Data (SPMD)**

Introduction - SPMD

- Basic idea:
  - *All logical processors (worker threads) execute the same program, with sequential code executed redundantly and parallel code (worksharing constructs, barriers, etc.) executed cooperatively*
- Exemplified by many popular parallel execution models
  - OpenMP for multicore systems
  - CUDA and OpenCL for accelerator systems
  - MPI for distributed systems

Introduction - Data races

- Data races are a pernicious source of bugs in SPMD model (Shared memory)
- Definition:
  - *In general, a data race occurs when two or more threads perform a conflicting accesses (at least one access being write) to a shared variable without any synchronization among threads.*
- Occurs only in few of the possible schedules of a parallel program
  - Extremely hard to reproduce and debug!

Motivation and Our Approach

- **Motivation**
  - Popular use of high-level constructs and directives for expressing parallelism in source programs than low level constructs.
- **Our approach**
  - *Automatically detect data races in SPMD programs at compile time*

SPMD Parallelism using OpenMP

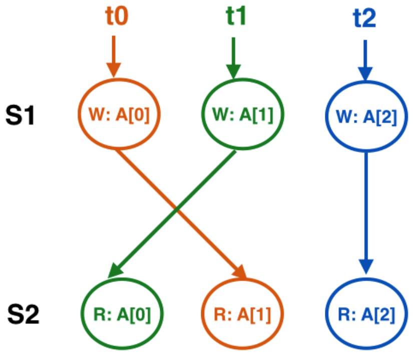
- Currently, we support following constructs in SPMD model
- OpenMP `parallel` construct
  - Creation of worker threads to execute an SPMD parallel region
- OpenMP `barrier` construct
  - Barrier operation among all threads in the current parallel region
  - Currently, we consider textually aligned barriers in SPMD region
- OpenMP `for` construct
  - Immediately following loop can be parallelized
  - Executed in a work-sharing mode by all the threads in the SPMD
    - **Schedule(static)**: Iterations are statically mapped to threads
    - **Schedule(dynamic)**: Iterations are dynamically mapped to threads.
  - **nowait** clause disables implicit barrier at end of the loop

Motivating example 1 - Any data race ??

- SPMD kernel with worksharing constructs

```
// tid - Thread id
// T - Total number of threads
#pragma omp parallel  shared(A) {
 #pragma omp for schedule(dynamic,1) nowait
     for(int i = 0; i < N; i++) {
          A[i] =  ...  // S1
     }

 #pragma omp for schedule(dynamic,1)
     for(int j = 0; j < N; j++) {
          ... = A[j]  // S2
     }
}
```



$N = 3, T = 3$

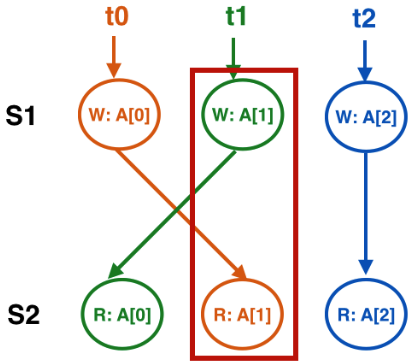
Motivating example 1 - Race b/w S1 and S2

- SPMD kernel with worksharing constructs

```
// tid - Thread id
// T - Total number of threads
#pragma omp parallel  shared(A) {
 #pragma omp for schedule(dynamic,1) nowait
    for(int i = 0; i < N; i++) {
        A[i] =  ...  // S1
    }

 #pragma omp for schedule(dynamic,1)
    for(int j = 0; j < N; j++) {
        ... = A[j]  // S2
    }
}
```

- Race between read of A[i] in S1 ($i = 1$) and write to A[i] in S2 ($i = 1$)

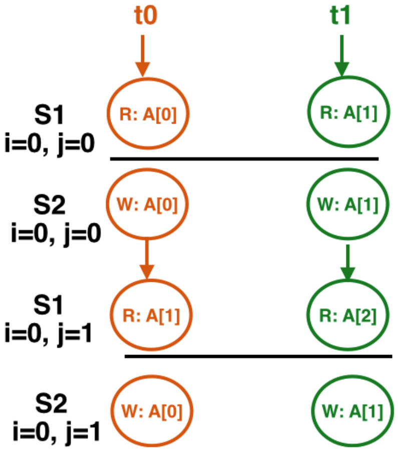


$N = 3, T = 3$

Motivating example 2 - Any data race ??

- SPMD kernel with barriers

```
// tid - Thread id
// T - Total number of threads
#pragma omp parallel  shared(A) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            int temp = A[tid + i + j];//S1
            #pragma omp barrier
            A[tid] += temp; //S2
        }
    }
}
```



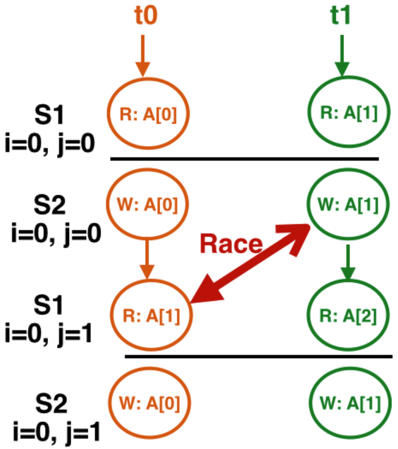
$T = 2$

Motivating example 2 - Race b/w S1 and S2

- SPMD kernel with barriers

```
// tid - Thread id
// T - Total number of threads
#pragma omp parallel  shared(A) {
     for(int i = 0; i < N; i++) {
          for(int j = 0; j < N; j++) {
               int temp = A[tid + i + j];//S1
               #pragma omp barrier
               A[tid] += temp; //S2
          }
     }
}
```

- Race between read of A[tid+ i + j] in S1 (tid = 0, i = 0, j = 1) and write of A[tid] in S2 (tid = 1, i = 0, j = 0)



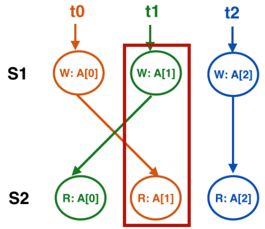
$T = 2$

Our Contributions

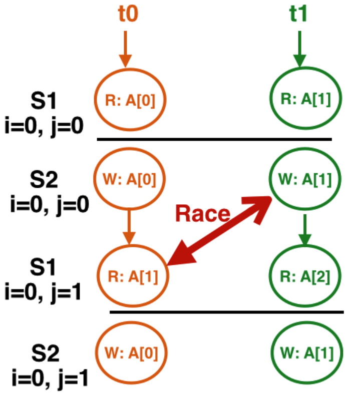
- Extensions to the polyhedral model for SPMD programs
- Formalization of May Happen in Parallel (MHP) relations in the extended model
- An approach for static data race detection in SPMD programs

May Happen in Parallel relation

- May Happen in Parallel relation
  - Specification of partial order among dynamic statement instances
  - MHP(S1, S2) = *true* $\leftrightarrow$ S1 happens in parallel with S2, where S1 and S2 are statement instances.



MHP(S1 (i = 1), S2(i = 1)) = true



MHP(S1 (tid=0, i=0, j=1), S2(tid=1,i=0, j=0)) = true

Z3 solver (Microsoft Research)

- SMT solver to check the satisfiability of logical formuale
- Output: sat/ un-sat/un-decidable
- If the logical formula is satisfiable from the solver, then there exists an assignment that marks logical formula as true

- Support for uninterpreted functions, non-linear arithmetic, divisions, quantifiers etc.

Polyhedral Compilation Techniques

- Compiler techniques for analysis and transformation of codes with nested loops
- Algebraic framework for affine program optimizations
- Advantages over AST based frameworks
    - Reasoning at statement instance level
    - Unifies many complex loop transformations

Polyhedral Representation (SCoP)

- A statement (S) in the program is represented as follows in Static Control Part (SCoP):
- 1) Iteration domain ($\mathcal{D}^{S}$)
  - Set of statement (S) instances
- 2) Scattering function (space-time mapping)
  - Space mapping: Allocation
    - Assigns logical thread ids to the statement instances (S)
  - Time mapping: Schedule (Θ^{S})
    - Assigns logical time stamps to the statement instances (S)
    - Gives ordering information b/w statement instances
    - **Captures sequential execution order of a program**
    - Statement instances are executed in increasing order of schedules
- 3) Access function ($\mathcal{A}^{S}$)
  - Array subscripts in the statement (S)

Can space-time mapping capture orderings in SPMD programs ?

- Major difference between Sequential and Parallel programs
    - Sequential programs - total execution order
    - Parallel programs - partial execution order
- Can Space-Time mapping (scattering function) capture all possible orderings in a given SPMD program?

Can space-time mapping capture orderings in SPMD programs ?

- Consider the following simpler example with a barrier

```
// tid - thread id
// T - total number of threads
#pragma omp parallel
    {
        S1;
        S2;
        #pragma omp barrier
        S3;
    }
```

space-time mapping:

S1: (tid, 0)

S2: (tid, 1)

S3: (tid, 2)

- Does this scattering function capture all possible orderings ??
- Captures ordering within a thread
- But, It doesn't capture ordering across threads (E.g: Barriers)

Can space-time mapping capture orderings in SPMD programs ?

- Consider the following simpler example with a barrier

```
// tid - thread id
// T - total number of threads
#pragma omp parallel
    {
        S1;
        S2;
        #pragma omp barrier
        S3;
    }
```

space-time mapping:

S1: (tid, 0)

S2: (tid, 1)

S3: (tid, 2)

- Does this scattering function capture all possible orderings ??
- Captures ordering within a thread
- But, It doesn't capture ordering across threads (E.g: Barriers)

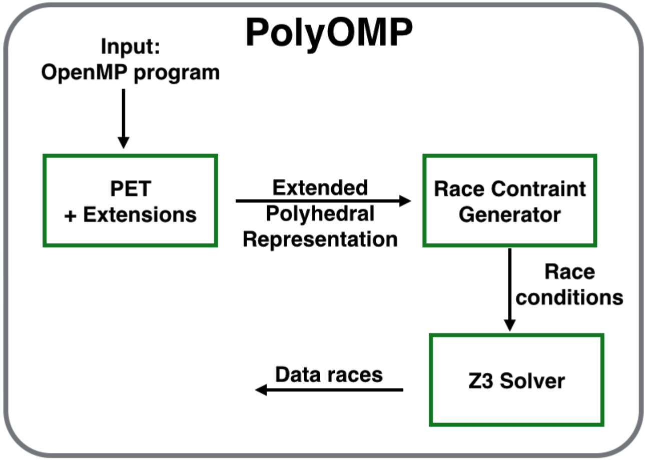
Polyhedral Compilation Techniques - Summary

- Advantages
  - Precise data dependency computation
  - Unified formulation of complex set of loop transformations
- Limitations
  - Affine array subscripts
    - But, conservative approaches exist !
  - Static affine control flow
    - Control dependences are modeled in same way as data dependences.
  - **Assumes input is sequential program**
    - **Unaware of all possible orderings in input parallel program**

Intuition behind Data race detection algorithm

- In order to check for race at static time b/w stmt instances S and T,
  - Generate race condition between S and T as follows
    - *S and T may touch same memory location, at least one of which is write*
    - *S and T may happen in parallel*
  - Forward the race condition to Z3 SMT solver
  - If the race condition is unsatisfiable, then there is NO race (Assuming no-aliasing)
  - If the race condition is satisfiable, then there MAY be a race
    - If there are no conservative estimations used during representation, then it is a PRECISE race.

Our workflow



Extended Polyhedral Representation

- Introduced **Phase** mapping to the scattering function (Space-Time mapping)
- Phase mapping:
  - **Motivation**: SPMD program execution can be partitioned into a sequence of phases separated by barriers.
  - Assigns a logical identifier, that we refer to as a *phase stamp*, to each statement instance. (Can be multi-dimensional like schedules)
  - Statement instances are executed according to increasing lexicographic order of their phase-stamps
- Now, Scattering function = Space-Phase-Time mapping

Reachable barriers of stmt instance S?

- **Defn**: Reachable barriers of a stmt instance S
  - Set of barrier instances that may be executed after S without an intervening barrier. (Similar to reachable definitions)

- SPMD kernel with barriers

```
// tid - Thread id
// T - Total number of threads
#pragma omp parallel  shared(A) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            int temp = A[tid + i + j];//S1
            #pragma omp barrier // B
            A[tid] += temp; //S2
        }
    }
}
```

Reachable barriers of S1 (i, j):
- B(i, j)

Reachable barriers of stmt instance S?

- **Defn**: Reachable barriers of a stmt instance S
  - Set of barrier instances that may be executed after S without an intervening barrier. (Similar to reachable definitions)

- SPMD kernel with barriers

```
// tid - Thread id
// T - Total number of threads
#pragma omp parallel shared(A) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            int temp = A[tid + i + j];//S1
            #pragma omp barrier // B
            A[tid] += temp; //S2
        }
    }
}
```

Reachable barriers of S2 (i, j):
- B(i, j+1) if $j < N-1$
- B(i+1, 0) if $j = N-1$

How to compute Phase mapping of S?

- How to compute Phase mapping of S ?
  - Treat barriers also as regular statement
  - Compute 2d+1 regular schedules for all statements
  - Phase mapping of S = OR of time-mappings of barriers in Reachable barriers of S

- SPMD kernel with barriers

```
// tid - Thread id
// T - Total number of threads
#pragma omp parallel  shared(A) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            int temp = A[tid + i + j];//S1
            #pragma omp barrier // B
            A[tid] += temp; //S2
        }
    }
}
```

Reachable barriers of S1 (i, j):
- B(i, j)
Time mapping of B(i, j):
- (i, j, 1)
Phase mapping of S1 (i, j) :
- (i, j, 1)

How to compute Phase mapping of S?

- How to compute Phase mapping of S ?
  - Treat barriers also as regular statement
  - Compute 2d+1 regular schedules for all statements
  - Phase mapping of S = OR of time-mappings of barriers in Reachable barriers of S

- SPMD kernel with barriers

```
// tid - Thread id
// T - Total number of threads
#pragma omp parallel  shared(A) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            int temp = A[tid + i + j];//S1
            #pragma omp barrier // B
            A[tid] += temp; //S2
        }
    }
}
```

Reachable barriers of S2 (i, j):
- B(i, j+1) if j < N-1
- B(i+1, 0) if j = N-1

Time mapping of B(i, j):
- (i, j, 1)

Phase mapping of S2 (i, j) :
- (i, j+1, 1)
- (i+1, 0, 1)

How to compute May Happen in Parallel (MHP) relations?

- In general, two stmt instances S and T in a parallel region can be run in parallel if and only if both of them are in same phase of computation (not ordered by synchronization) and are executed by different threads in the region.
- MHP(S, T) is true iff
  - *Executed by different threads, Space(S) != Space(T)*
  - *And Same execution phase, Phase(S) = Phase(T)*

Race detection: Step - 1 : Generated race condition

-
  ```
  // tid - Thread id, T - Total number of threads
  #pragma omp parallel  shared(A) {
      for(int i = 0; i < N; i++) {
          for(int j = 0; j < N; j++) {
              int temp = A[tid + i + j];//S1
             #pragma omp barrier
             A[tid] += temp; //S2
          }
      }
  }
  ```
- Race condition b/w S1(tid$_{S1}$, i, j) and S2(tid$_{S2}$, i′, j′):

Same access:$(0 \leq i, j < N) \wedge (0 \leq i', j' < N) \wedge (tid_{S1} + i + j = tid_{S2})$

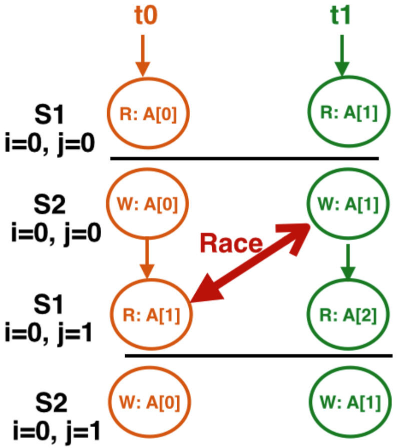
Different threads: $\wedge\ (tid_{S1} \neq tid_{S2})$

Same phase: $\wedge\ ((i = i' + 1 \wedge j = 0 \wedge j' = N - 1) \vee (i = i' \wedge j = j' + 1 \wedge j' < N - 1))$

Race detection: Step - 2 : Z3 solver

```
// tid - Thread id
// T - Total number of threads
#pragma omp parallel  shared(A) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            int temp = A[tid + i + j];//S1
            #pragma omp barrier
            A[tid] += temp; //S2
        }
    }
}
```

- Satisfiable assignment from Z3 solver:
- S1(tid_{S1} = 0, i = 0, j=1)
- S2(tid_{S2} = 1, i = 0, j=0)



Assumptions/ Limitations

- We currently support textually aligned barriers
    - Hard to identify which barriers (unaligned) form a synchronization point
- We assume no aliasing on variables
    - Can be supported with aliasing analysis done before race analysis
- Generated race conditions are decidable.
    - This is true in case of basic constructs such as worksharing and barriers.
- Support for only dynamic schedule in the worksharing construct
    - Static schedule with variable chunk size introduces non-affine terms.

Related work

	Supported Constructs	**Approach**	**Guarantees**	**False +Ves**	**False -Ves**
Pathg (Yu et.al) LCTES'12	OpenMP worksharing loops, Barriers, Atomic	Thread automata	Per number of threads	Yes	No
OAT (Ma et.al) ICPP'13	OpenMP worksharing loops, Barriers, locks, Atomic, single, master	Symbolic execution	Per number of threads	Yes	No
ompVerify (Basupalli et.al) IWOMP'11	OpenMP 'parallel for'	Polyhedral (Dependence analysis)	Per 'parallel for' loop	No - (Affine subscripts)	No - (Affine subscripts)
polyX10 (Yuki et.al) PPoPP'13	X10 Async/ finish	Polyhedral (HB relations)	Per a captured SCoP	No - (Affine subscripts)	No - (Affine subscripts)
PolyOMP (Chatarasi et.al) IMPACT'16	OpenMP worksharing loops, Barriers, Single, master	Polyhedral (MHP relations)	Per SPMD region	No - (Affine subscripts) Yes - (Non affine)	No

PolyOMP - Conclusions and Future work

- Conclusions:
  - Extensions to the polyhedral model for SPMD programs
  - Formalization of May Happen in Parallel (MHP) relations in the extended model
  - An approach for static data race detection in SPMD programs
- Future work:
  - Support for textually unaligned barriers
  - Extend the analysis for more constructs such as doacross etc.
  - Transformations of SPMD regions such as SPMD fusion

Finally,

- *Representing explicitly parallel programs in polyhedral model is a new direction for both analysis and transformations of parallel programs!*
- Acknowledgments
  - IMPACT 2016 Program Committee
  - Rice Habanero Extreme Scale Software Research Group
- Thank you!

Related work

- Extensive body of literature on identifying races in explicitly parallel programs
- Symbolic approaches in the analysis of OpenMP programs
  - Consistency checking using thread automata [Yu et.al LCTES'12]
    - Support for guided witness search to show inconsistency
    - Guaranteed only for fixed number of threads
  - Analysis of concurrency errors [Ma et.al ICPP'13]
    - Support for guided witness search to show inconsistency
    - Guaranteed only for fixed number of threads
  - We extend the polyhedral framework to generate constraints that can be applicable to any variable number of worker threads

Related work (contd)

- Polyhedral approaches
  - Polyhedral optimizations of explicitly parallel programs [Chatarasi et.al PACT'15]
    - Analyze and Optimize parallel loops, task constructs in OpenMP
    - Supports only parallel constructs satisfying `serial-elision` property
  - Polyhedral approaches for OpenMP race detection [Basupalli et.al IWOMP'11]
    - Converts C-Programs into standard polyhedral representation
    - Supports only `omp parallel for`
  - Polyhedral approaches for X10 race detection [Yuki et.al PPoPP'13]
    - HB relation are analyzed and data-flow is computed and data-flow is computed based on partial order imposed by HB relation.
    - This data-flow is used to certify determinacy.
    - Supports only `async` and `finish` constructs
  - Our approach is applicable to parallel SPMD programs in general