Hybrid Analysis of SystemC Models for Fast and Accurate Parallel Simulation

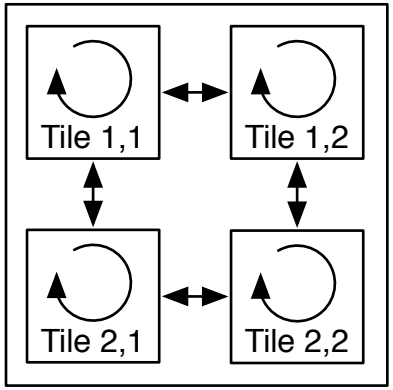
Tim Schmidt, Guantao Liu, and Rainer Dömer
Center for Embedded and Cyber-physical Systems
University of California, Irvine, USA

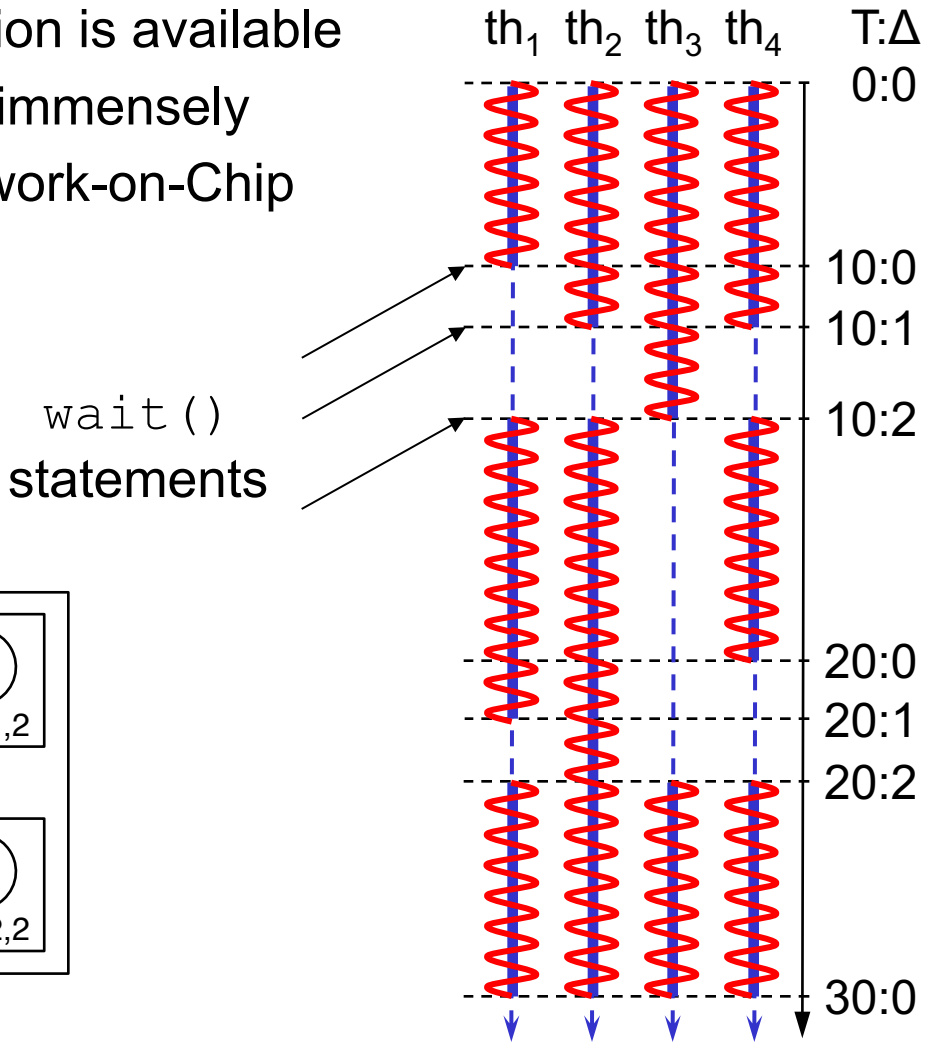
UCIrvine
University of California, Irvine

Traditional Parallel Simulation

- Parallel SystemC simulation is available
- Reduces simulation time immensely
- Highly applicable for Network-on-Chip applications

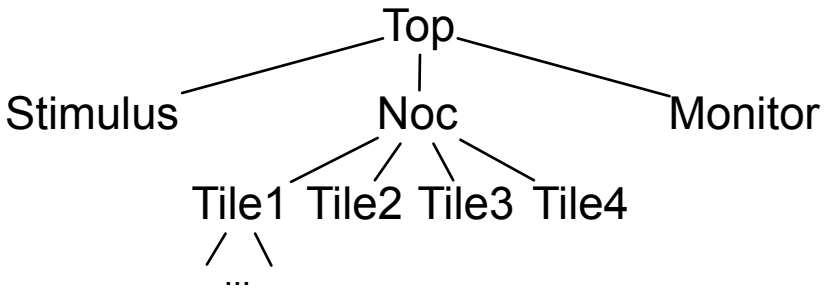
- Example:





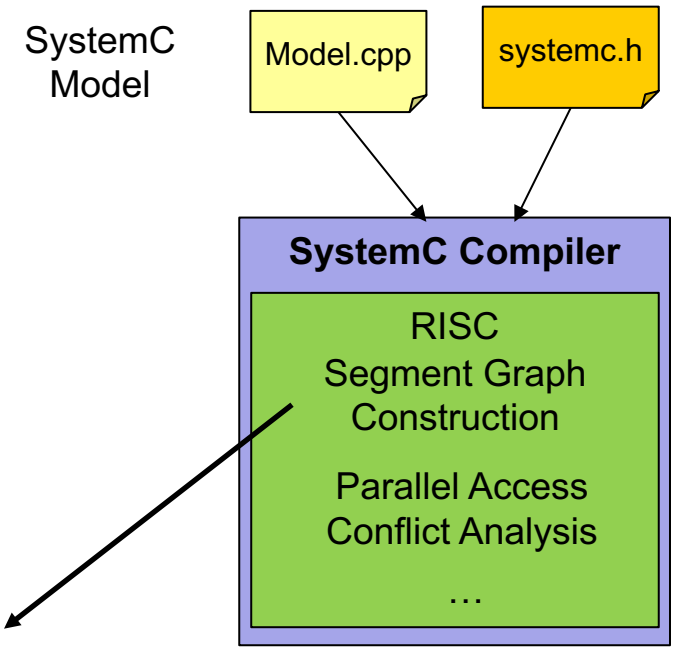
Traditional Parallel Simulation

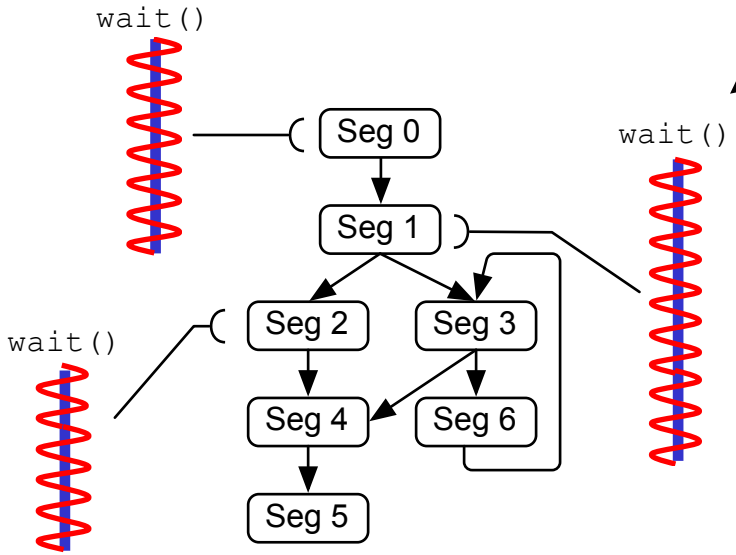
- Compiler driven strategy for parallelization
  1. Identify the module hierarchy



Traditional Parallel Simulation

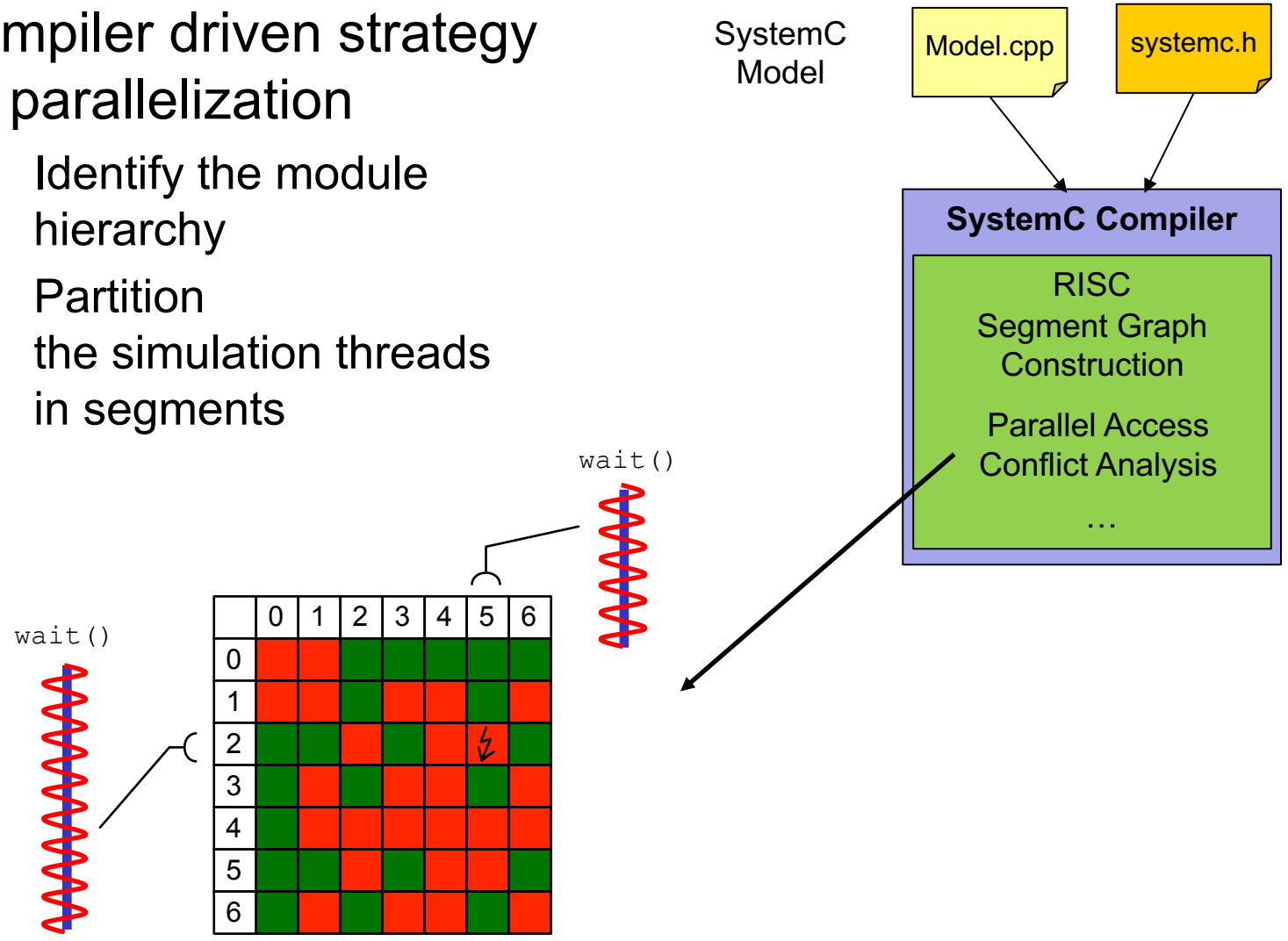
- Compiler driven strategy for parallelization
  1. Identify the module hierarchy
  2. Partition the simulation threads in segments





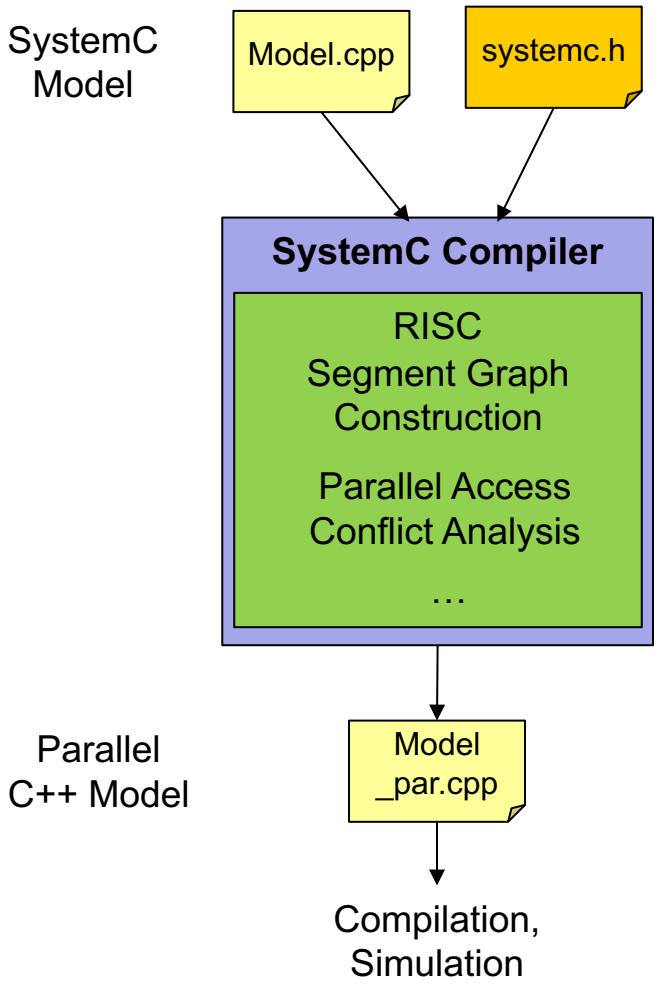
Traditional Parallel Simulation

- Compiler driven strategy for parallelization
  1. Identify the module hierarchy
  2. Partition the simulation threads in segments



Traditional Parallel Simulation

- Compiler driven strategy for parallelization
  1. Identify the module hierarchy
  2. Partition the simulation threads in segments



Traditional Parallel Simulation

- Compiler driven strategy for parallelization
  1. Identify the module hierarchy
  2. Partition the simulation threads in segments
  3. Instrument source code for the parallel simulator

C++ Model

```
int conflict_table[x][y];

...

void thread()
{
   ...
   wait(... , Segment ID);
   ...
}

...
```

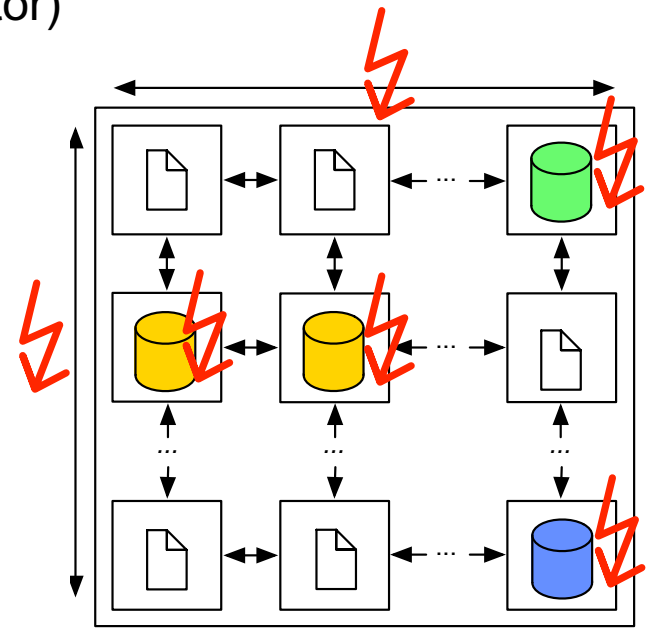
Outline

- Problem Definition
- Hybrid Analysis
- Library Handling
- Experiments
- Conclusion

Problem Definition

Parallel simulation requires static analysis

1. Designs with libraries cannot be analyzed (source code needed)
2. Dynamic resizable designs cannot be considered (`new` operator)



Basic idea:

- We replace static analysis with **hybrid analysis**
- We introduce dedicated **library handling**

Hybrid Analysis

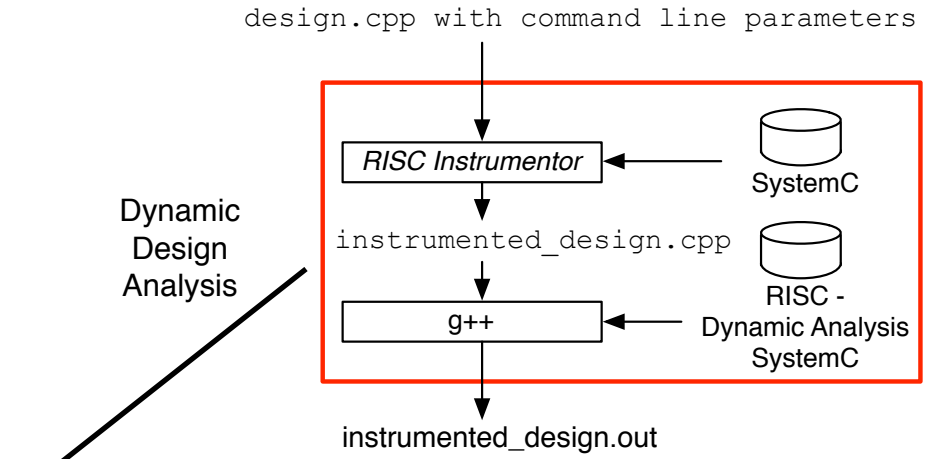
- Hybrid Analysis = Dynamic Analysis + Static Analysis

- Dynamic Analysis
  - Simulate design until evaluation phase
  - Write design structure in a file

- Static Analysis
  - Perform regular static analysis
  - Use dynamic analysis results to support static

Hybrid Analysis

1. Instrumented design for *Dynamic Design Analysis*



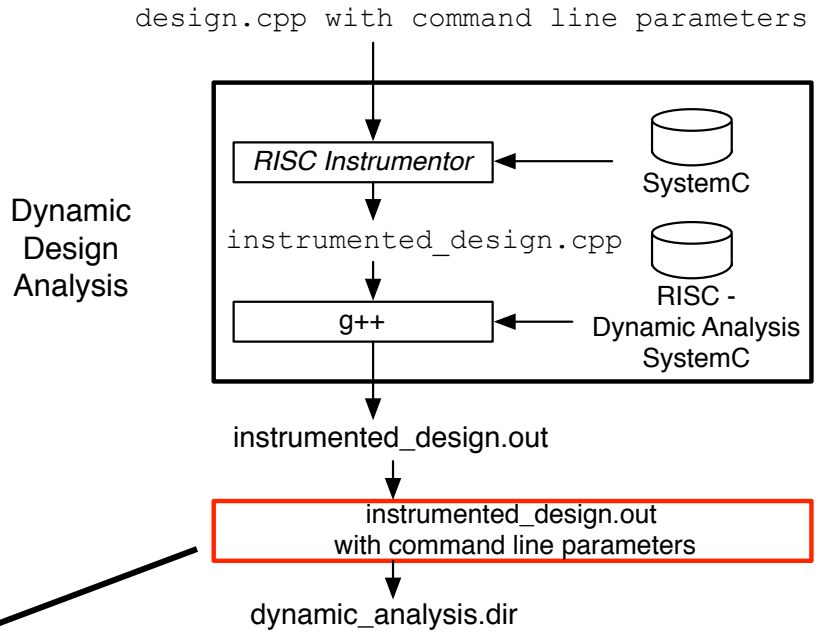
Dynamic Design Analysis

instrumented_design.cpp

```
...

void print_variables(FILE *file)
{
  fprintf(file,
          "reference_@%p,",
          &reference_);

  fprintf(file,
          "variable_$%p,",
          &variable_);
}

void print_ports(FILE *file)
{ ... }

...
```

Hybrid Analysis

1. Instrumented design for *Dynamic Design Analysis*
2. Execute instrumented design

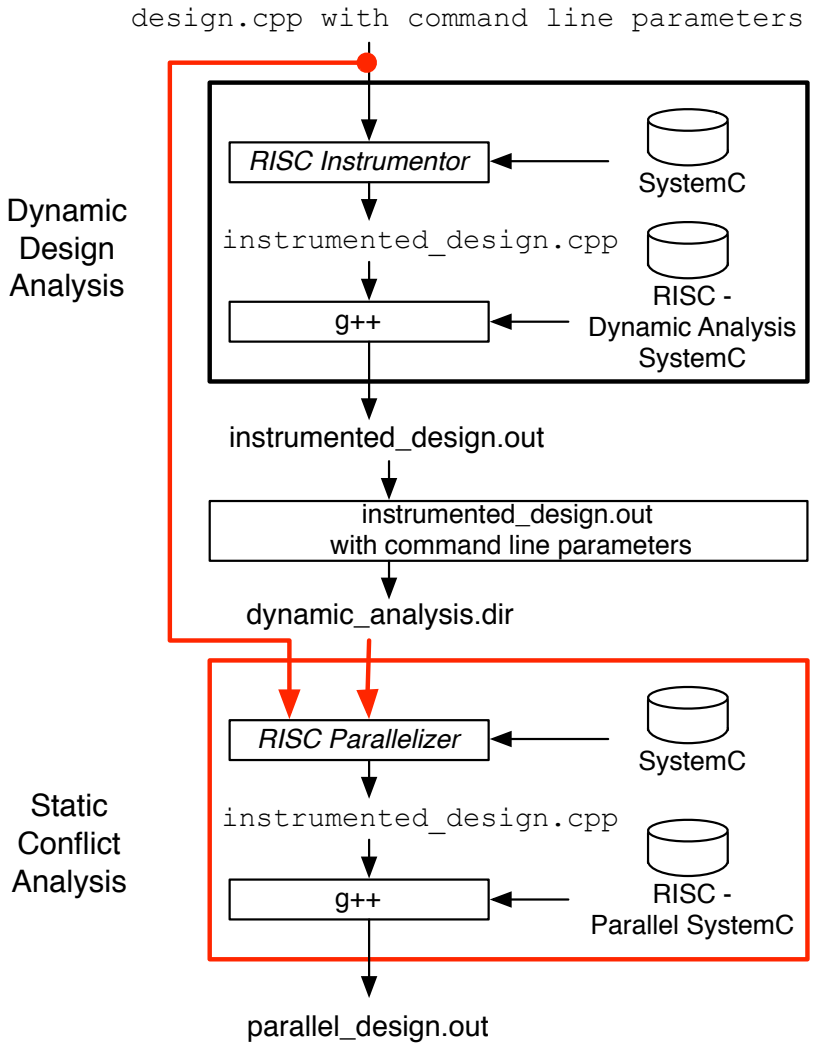


dynamic_analysis.dir

```
top:0x111(
  chnl1:0x222:MyChannelType,
  var:0x333 mod1:0x444(
    var:0x555,
    ref:0x333,port:0x222))
```

Hybrid Analysis

1. Instrumented design for *Dynamic Design Analysis*
2. Execute instrumented design
3. Perform *Static Conflict Analysis*



Library Handling

- Libraries provide only interfaces, no source code
- Only header files are available

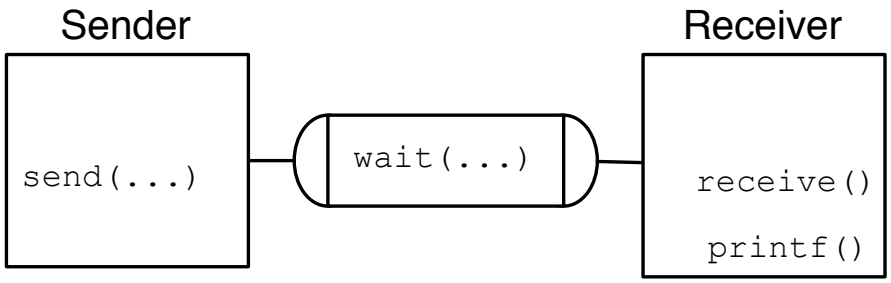
- Static analysis cannot identify `wait` statements in libraries
- Static analysis cannot instrument `wait` statements in libraries

- We provide annotation scheme for functions
  Example:

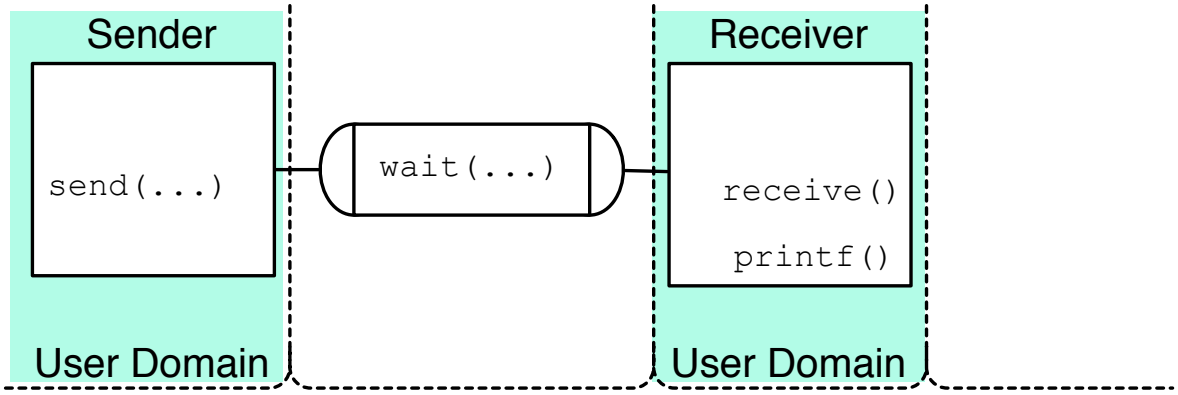
```
#pragma RISC no-wait
void foo();
```

How can we pass information to the simulator
without modify the library?

Library Handing

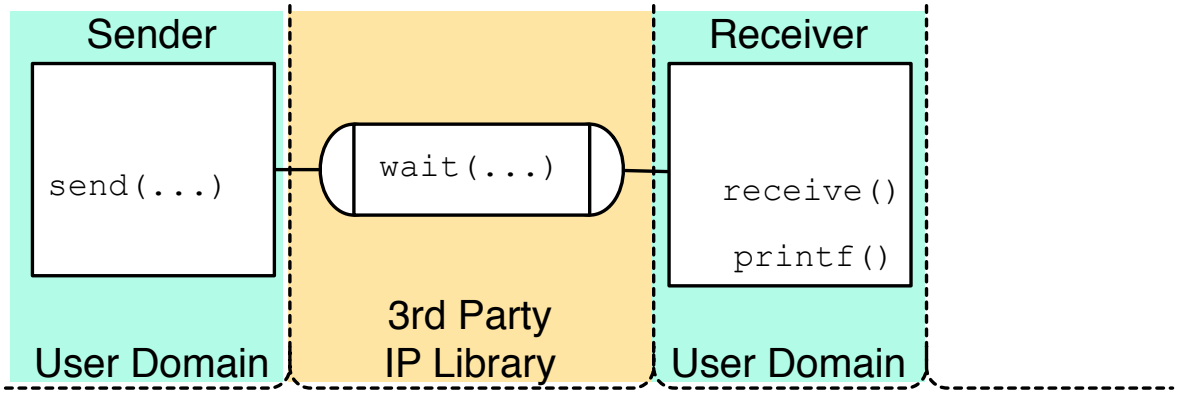


Library Handing



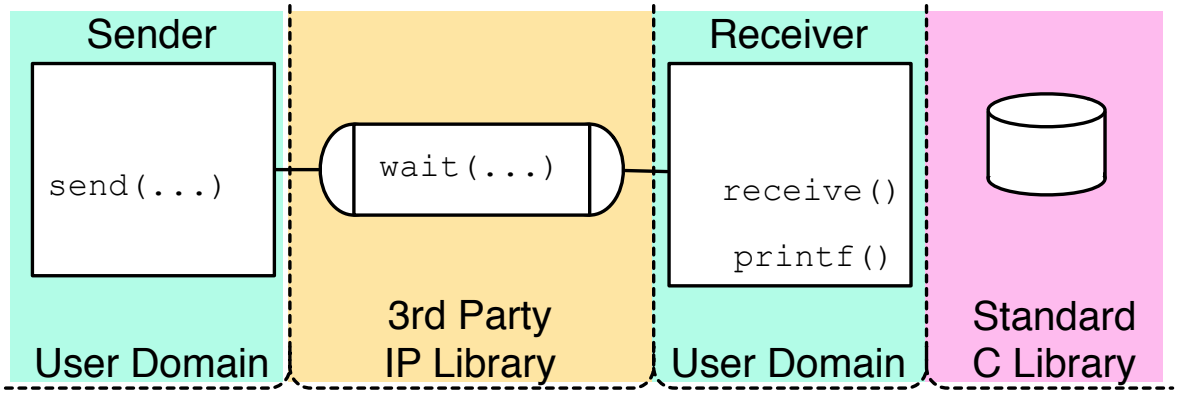
- User domain source code is available

Library Handing



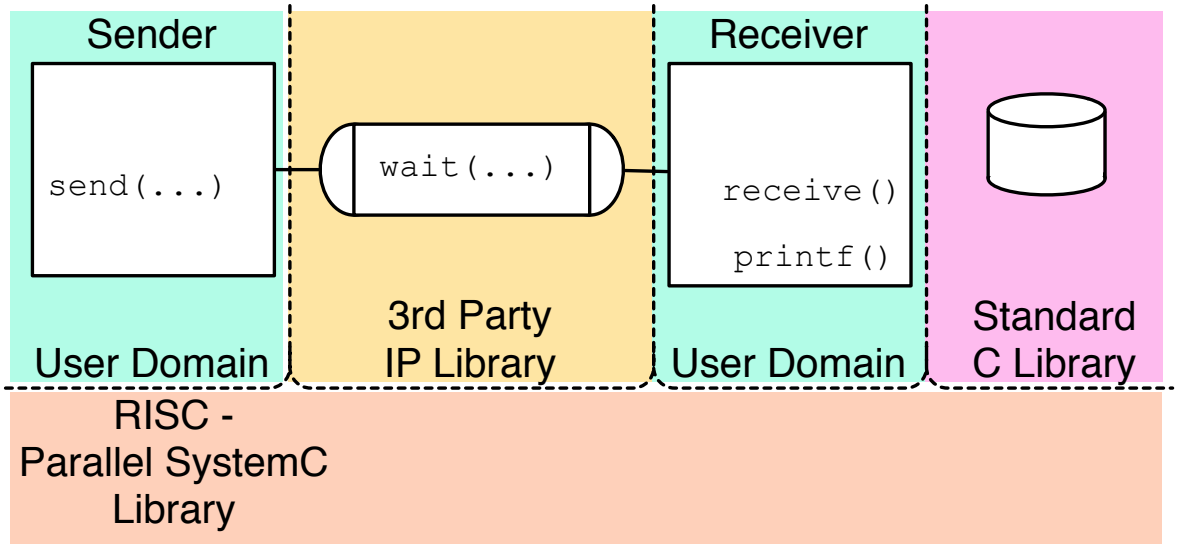
- 3rd party libraries provide only interfaces
- Changes and instrumentation is not possible

Library Handing



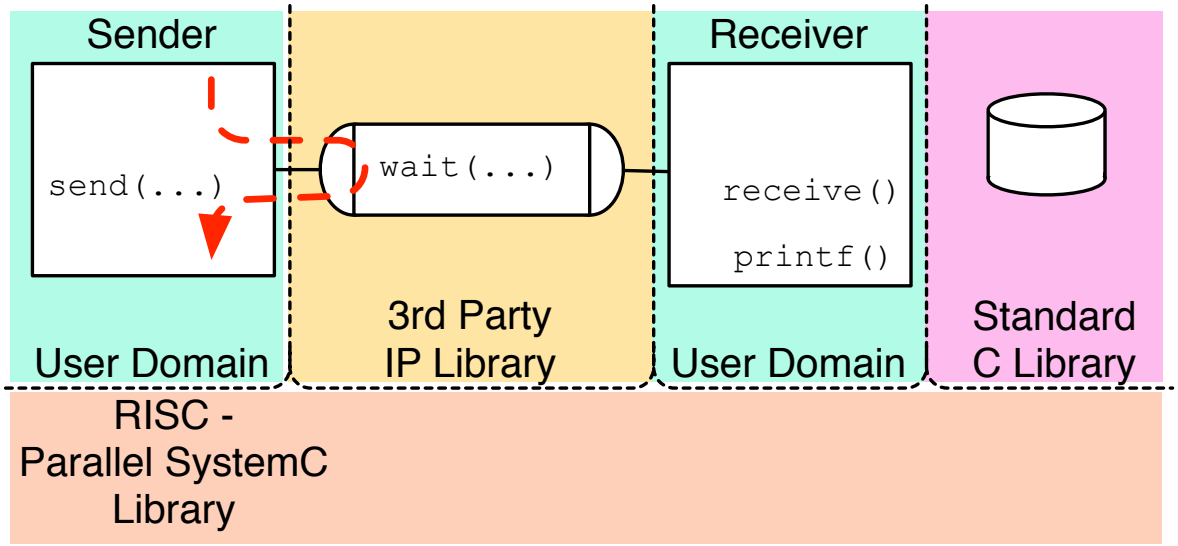
- Standard libraries provide only interfaces
- Changes and instrumentation is not possible

Library Handing

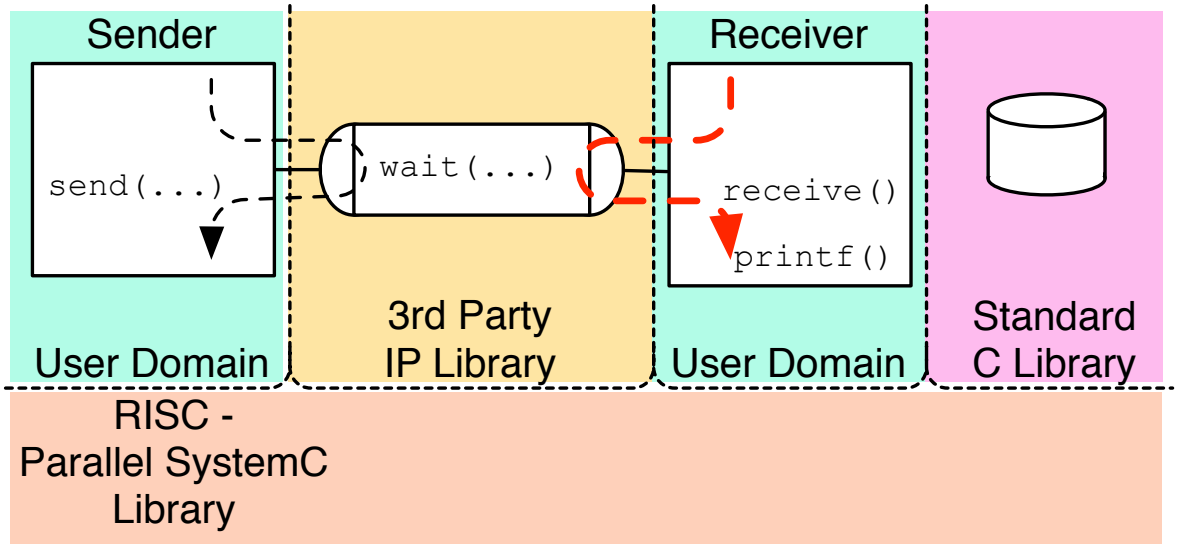


- RISC simulator with support for parallel SystemC

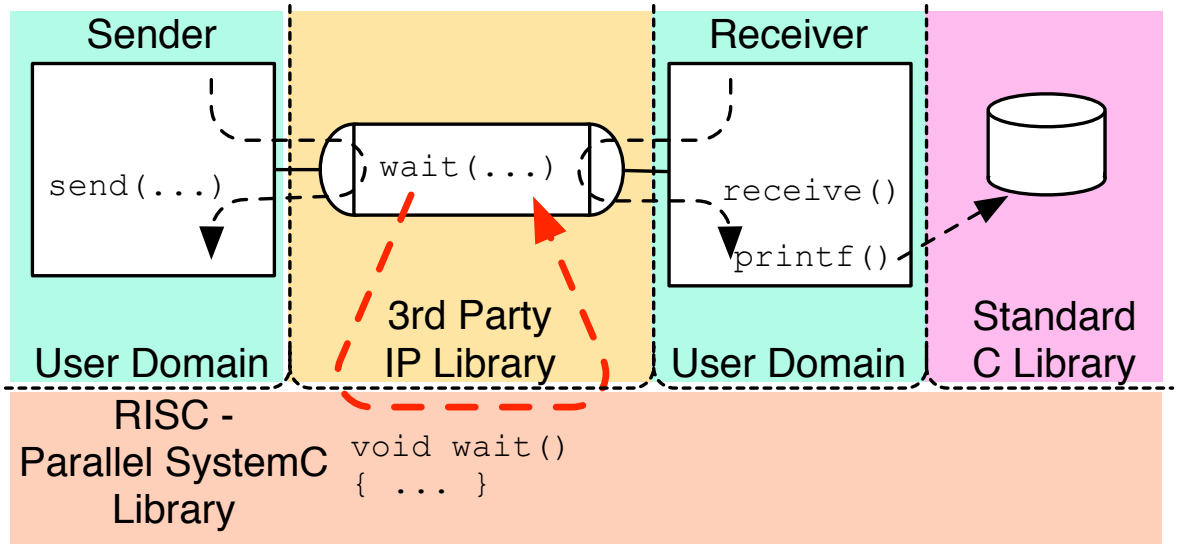
Library Handing



Library Handing

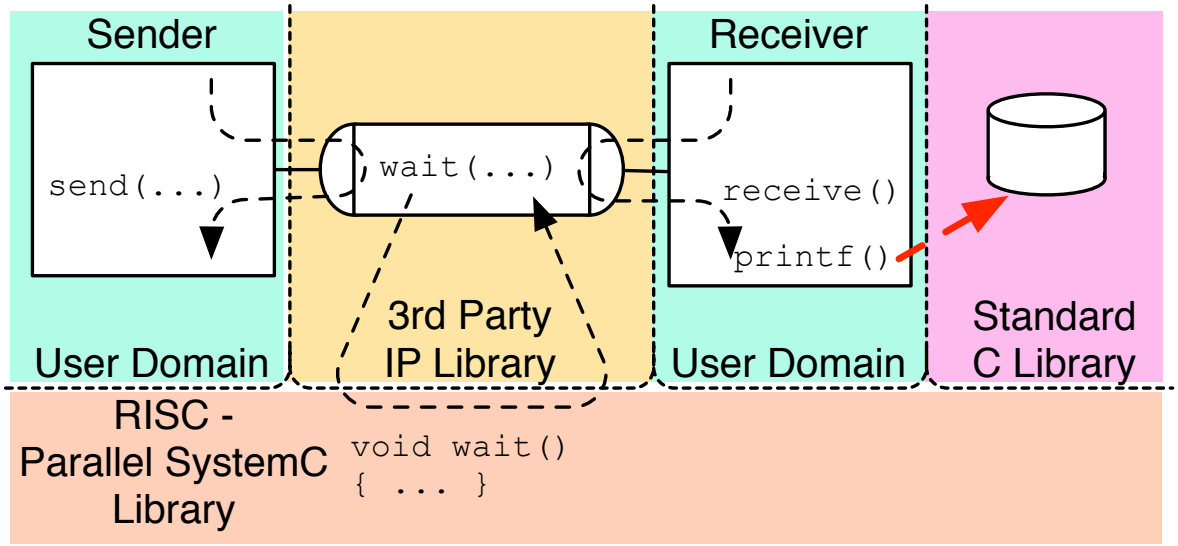


Library Handing

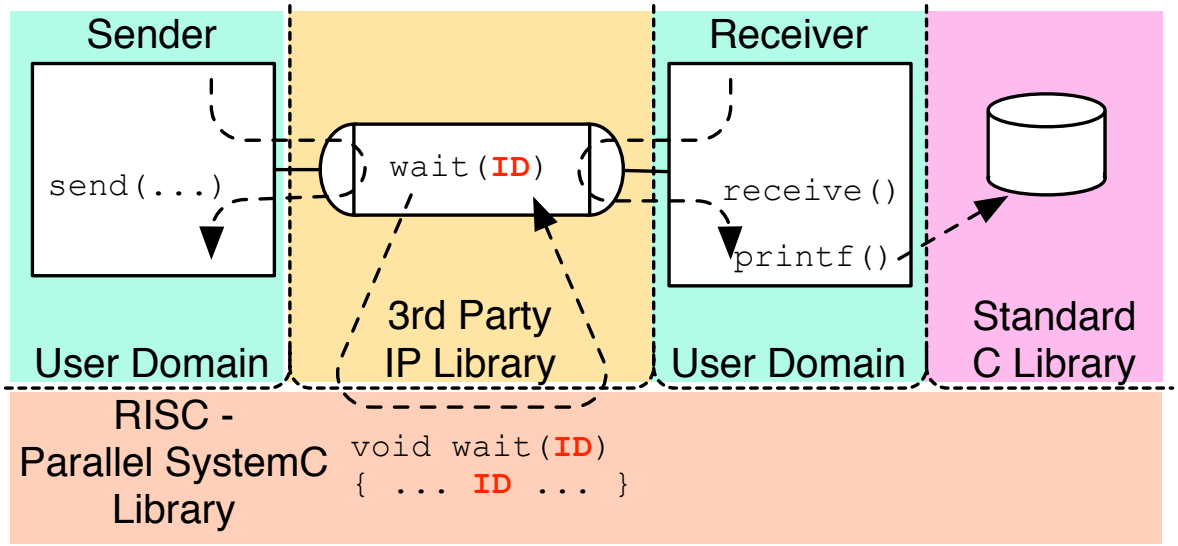


- `wait()` calls are synchronization points between the model and the simulator

Library Handing

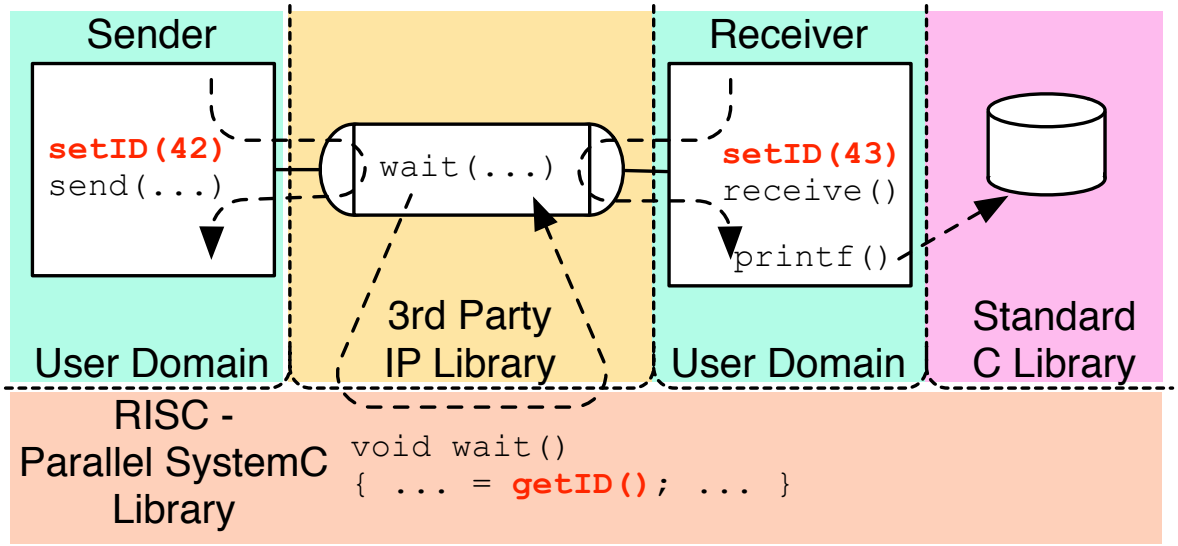


Library Handing



- RISC needs the segment ID to schedule the next segment
- We cannot instrument library code

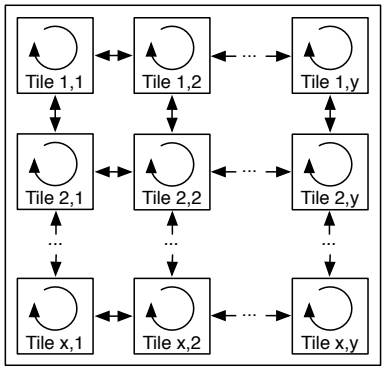
Library Handing



- We attach the upcoming segment ID with the thread local data
- We instrument `setID()` and `getID()`
- `wait()` calls do not change anymore

Experiments

- Simulation of a Network-on-Chip particle simulator
  - 65 modules
  - 176 channels
  - # cores 5x5 → 8x8
  - # particles 10k → 60k



- Simulation Host:
  Intel Xeon E3-1240 with 4 cores and 2 threads per core

- Simulation results 100% accurate with sequential simulation

	Time (in sec.)		Speedup			
5x5	160.20	53.77	2.56x	3.58x	3.25x	2.97x
6x6	126.53	29.09	2.80x	4.88x	5.04x	4.34x
7x7	117.46	24.11	2.32x	3.91x	5.01x	4.87x
8x8	108.40	16.96	2.07x	4.12x	6.05x	6.39x
Particles	seq. 60k	par. 60k	10k	20k	40k	60k

Conclusion

- Traditional Parallel Discrete Event Simulation has limitations
- We propose
  - Hybrid analysis of models
  - Library support for parallel simulation
- Our experiments
  - NoC particle simulator
  - Flexible number of cores 5x5 → 8x8
  - Maintaining 100% accuracy
  - Speedup up to 6.39x