Automatic Verification of Determinism for Structured Parallelism

Martin Vechev

IBM T.J. Watson Research Center

joint work with:

Eran Yahav (Technion), Raghavan Raman, Vivek Sarkar (Rice)

Outline

- Motivation
- Checking: concrete (finite) reasoning
- Proving: abstract (infinite) reasoning
- Experimental results
- Limitations

Motivation

- Increasing number of Parallel Algorithms
  - Scientific Computing
  - Signal Processing
  - Encryption
  - Sorting
  - Searching
  - String Indexing
- Meant to be deterministic
  - for the same input, must produce the same output

Structured Concurrency

- Structured parallel languages
  - Renewed interest: FJ (Java), Cilk (MIT), X10 (IBM), DPJ (UIUC), TPL (Microsoft), HJ (Rice), CUDA (NVIDIA)
  - Can capture many parallel algorithms
- Benefits of structured languages
  - Deadlock-freedom (certain kinds)
  - Concurrency Analyses (better complexity)
- We assume a simple async-finish language
  - "FX10: a core calculus for async-finish parallelism" Lee & Palsberg (PPoPP'2010)

Motivating Example

```
void update (double[][] G, int start, int last, double c1, double c2, int nm, int ps) {
  finish {
    for (tid = start ; tid < last ; tid += 1) {
      async {
        int i = 2 * tid - ps;
        for (int j=1; j<nm; j++)
          G[i][j] = c1 * (G[i-1][j] + G[i+1][j] + G[i][j-1] + G[i][j+1]) + c2 * G[i][j];
      }
    }
  }
}
```

Uses parallel computation to apply the method of successive over-relaxation for solving a system of linear equations.

Motivating Example

```
void update (double[][] G, int start, int last, double c1, double c2, int nm, int ps) {
  finish {
    for (tid = start ; tid < last ; tid += 1) {
      async {
        int i = 2 * tid - ps;
        for (int j=1; j<nm; j++)
          G[i][j] = c1 * (G[i-1][j] + G[i+1][j] + G[i][j-1] + G[i][j+1]) + c2 * G[i][j];
      }
    }
  }
}
```

this is deterministic.
can we prove it ?

Uses parallel computation to apply the method of successive over-relaxation for solving a system of linear equations.

Determinism

proving arbitrary programs deterministic is hard

instead,

prove a stronger property which implies determinism

Outline

- Motivation
- Checking: concrete (finite) reasoning
- Proving: abstract (infinite) reasoning
- Experimental results
- Limitations

Conflict-Freedom

parallel tasks always access disjoint memory

Checking Conflict-Freedom

Step 1: Compute all reachable concrete states

Step 2: Check if each state is conflict-free

Conflicting State

A state where 2 tasks are enabled to access the same memory location and one of these accesses is a write

Conflict-Free Checker

For every program state σ

For all pairs of tasks t_1 and t_2 in σ:

if t_1 and t_2 are enabled and one is a write

if $Loc(t_1, \sigma) = Loc(t_2, \sigma)$

exit ("Program may be non-deterministic")

exit ("Program is deterministic")

Conflict-Free Checker

For every program state σ

For all pairs of tasks t_1 and t_2 in σ:

if t_1 and t_2 are enabled and one is a write

if $Loc(t_1, \sigma) = Loc(t_2, \sigma)$

exit ("Program may be non-deterministic")

temporal check

exit ("Program is deterministic")

Conflict-Free Checker

For every program state σ

For all pairs of tasks t_1 and t_2 in σ:

if t_1 and t_2 are enabled and one is a write

if $Loc(t_1, \sigma) = Loc(t_2, \sigma)$

exit ("Program may be non-deterministic")

exit ("Program is deterministic")

temporal check

spatial check

Example

```
void update (double[][] G, int start, int last, double c1, double c2, int nm, int ps) {
  finish {
    for (tid = start ; tid < last ; tid += 1) {
      async {
        int i = 2 * tid - ps;
        double[] Gi = G [i];
        double[] Gim = G [i - 1];
        double[] Gip = G [i + 1];
        for (int j=1; j < nm; j++)
          Gi[j] = c1 * (Gim[j] + Gip[j] + Gi[j-1] + Gi[j+1]) + c2 * Gi[j];
      }
    }
}}
```

Example

```
void update (double[][] G, int start, int last, double c1, double c2, int nm, int ps) {
  finish {
    for (tid = start ; tid < last ; tid += 1) {
      async {
        int i = 2 * tid - ps;
        double[] Gi = G [i];
        double[] Gim = G [i - 1];
        double[] Gip = G [i + 1];
        for (int j=1; j < nm; j++)
          Gi[j] = c1 * (Gim[j] + Gip[j] + Gi[j-1] + Gi[j+1]) + c2 * Gi[j];
      }
    }
}}
```

instantiate for 2 tasks with:

start = 1
last = 3
ps = 0
c1=c2=1
nm=2

Example (Instantiated)

```
void update (double[][] G) {
  finish {
    for (tid = 1 ; tid < 3 ; tid += 1) {
      async {
        int i = 2 * tid - 0;
        double[] Gi = G [i];
        double[] Gim = G [i - 1];
        double[] Gip = G [i + 1];
        for (int j=1; j < 2; j++)
          Gi[j] = (Gim[j] + Gip[j] + Gi[j-1] + Gi[j+1]) + Gi[j];
      }
    }
}}
```

Example: Conflict Free State

Task 1:

double[] Gi_1 = G[2];

double[] Gim_1 = G[1];

double[] Gip_1 = G[3];

tmp = Gim_1[1] + Gip_1[1] + Gi_1[0] +

Gi_1[2] + Gi_1[1];

Gi_1[1] = tmp;

Task 2:

double[] Gi_2 = G[4];

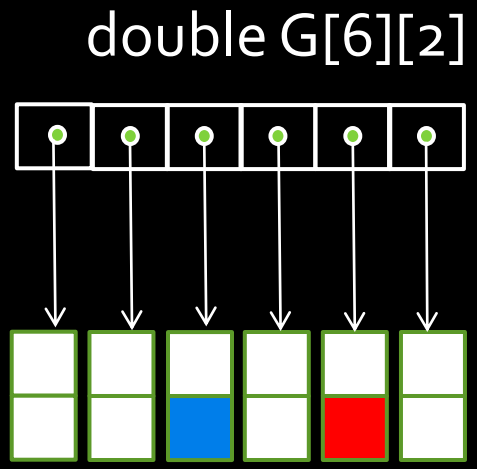
double[] Gim_2 = G[3];

double[] Gip_2 = G[5];

tmp = Gim_2[1] + Gip_2[1] + Gi_2[0] +

Gi_2[2] + Gi_2[1];

Gi_2[1] = tmp;



Example: Conflict Free State

Task 1:

double[] Gi_1 = G[2];

double[] Gim_1 = G[1];

double[] Gip_1 = G[3];

tmp = Gim_1[1] + Gip_1[1] + Gi_1[0] +

Gi_1[2] + Gi_1[1];

Gi_1[1] = tmp;

Task 2:

double[] Gi_2 = G[4];

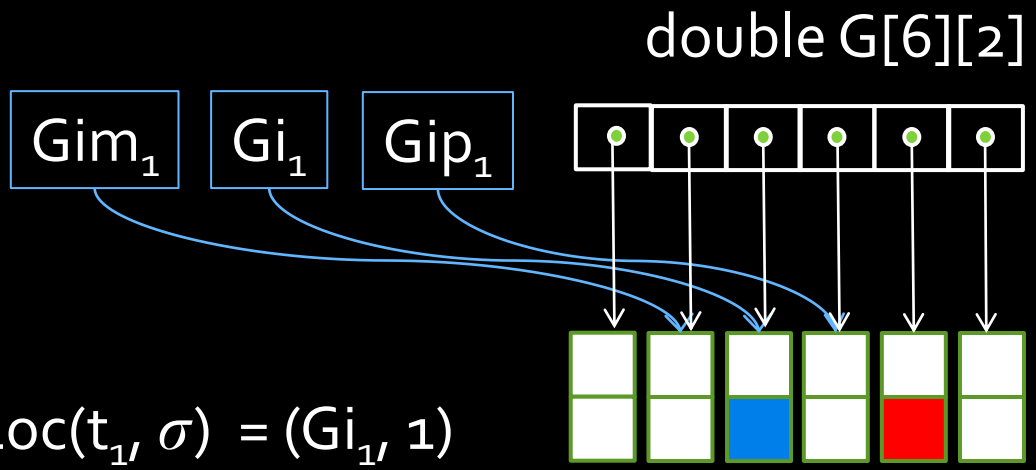
double[] Gim_2 = G[3];

double[] Gip_2 = G[5];

tmp = Gim_2[1] + Gip_2[1] + Gi_2[0] +

Gi_2[2] + Gi_2[1];

Gi_2[1] = tmp;



$Loc(t_1, \sigma) = (Gi_1, 1)$

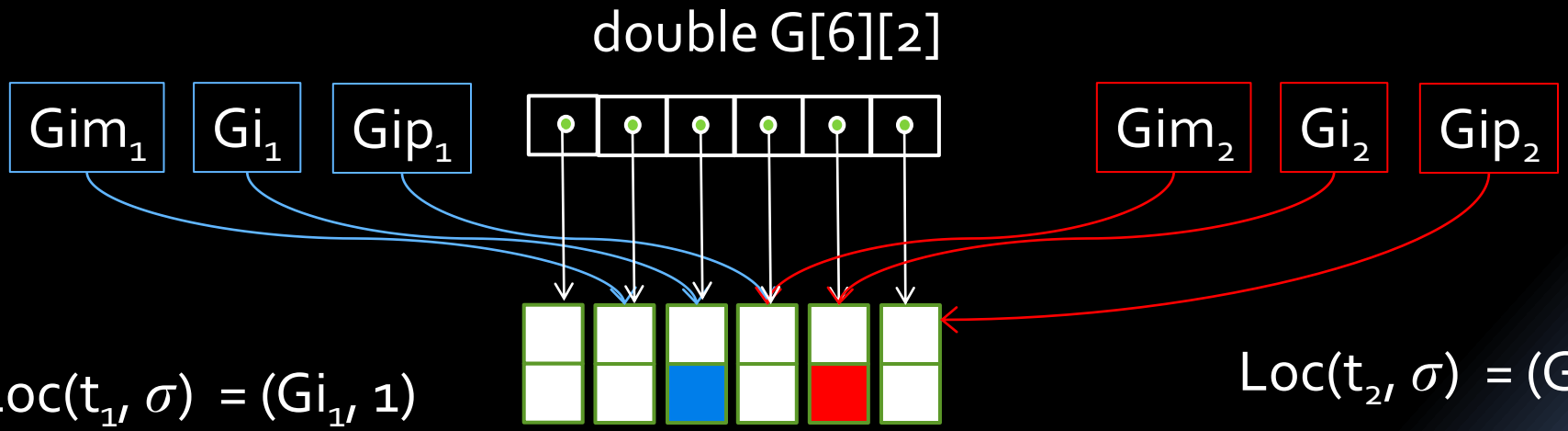
Example: Conflict Free State

Task 1:

```
double[] Gi1 = G[2];
double[] Gim1 = G[1];
double[] Gip1 = G[3];
tmp = Gim1[1] + Gip1[1] + Gi1[0] +
      Gi1[2] + Gi1[1];
Gi1[1] = tmp;
```

Task 2:

```
double[] Gi2 = G[4];
double[] Gim2 = G[3];
double[] Gip2 = G[5];
tmp = Gim2[1] + Gip2[1] + Gi2[0] +
      Gi2[2] + Gi2[1];
Gi2[1] = tmp;
```



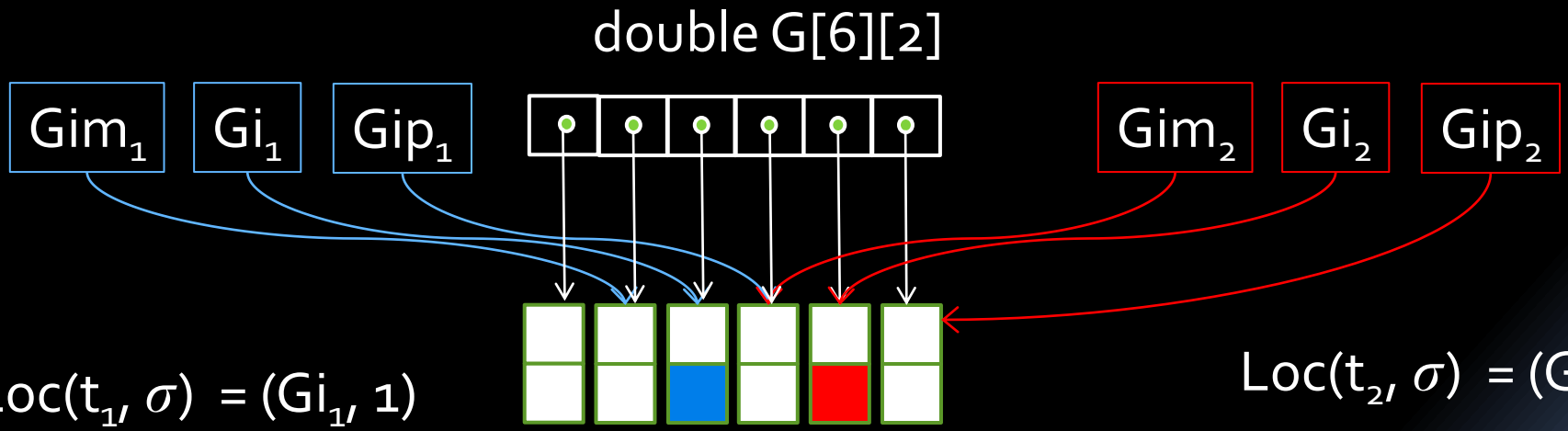
Example: Conflict Free State

Task 1:

```
double[] Gi1 = G[2];
double[] Gim1 = G[1];
double[] Gip1 = G[3];
tmp = Gim1[1] + Gip1[1] + Gi1[0] +
      Gi1[2] + Gi1[1];
Gi1[1] = tmp;
```

Task 2:

```
double[] Gi2 = G[4];
double[] Gim2 = G[3];
double[] Gip2 = G[5];
tmp = Gim2[1] + Gip2[1] + Gi2[0] +
      Gi2[2] + Gi2[1];
Gi2[1] = tmp;
```



Conflict-Free State ✓

Outline

- Motivation
- Checking: concrete (finite) reasoning
- Proving: abstract (infinite) reasoning
- Experimental results
- Limitations

Checking Conflict-Freedom

Step 1: Compute all reachable concrete states

Step 2: Check if the state is conflict-free

Checking Conflict-Freedom

Step 1: Compute all reachable concrete states

Bad News

Potentially Unbounded

Step 2: Check if the state is conflict-free

Sources of Unboundedness

- Unbounded Heap
- Unbounded range of array indices
- Unbounded number of tasks
  - Spawned in a loop

Dealing with Unboundedness

Use abstraction

Finite representation of an infinite number of states

Dealing with Unboundedness

- Unbounded Heap
- Unbounded range of array indices
- Unbounded number of tasks

Dealing with Unboundedness

- Unbounded Heap
  - points-to analysis: compute finite set of abstract locations
- Unbounded range of array indices
- Unbounded number of tasks

Dealing with Unboundedness

- Unbounded Heap
  - points-to analysis: compute finite set of abstract locations
- Unbounded range of array indices
  - numerical abstractions: compute symbolic constraints
- Unbounded number of tasks

Dealing with Unboundedness

- Unbounded Heap
  - points-to analysis: compute finite set of abstract locations
- Unbounded range of array indices
  - numerical abstractions: compute symbolic constraints
- Unbounded number of tasks
  - Compute invariants per task type: bounded number of task types (all tasks with same code)

A word on Numerical Domains

A word on Numerical Domains

- Polyhedra Domain [Cousot & Halbwachs, POPL'78]

A word on Numerical Domains

- Polyhedra Domain [Cousot & Halbwachs, POPL'78]
  - $a_1 x_1 + \ldots + a_n x_n \leq a_{n+1}$

A word on Numerical Domains

- Polyhedra Domain [Cousot & Halbwachs, POPL'78]
  - $a_1 x_1 + \ldots + a_n x_n \leq a_{n+1}$
  - $2x + y \leq 5$, but not $xy \leq 5$

A word on Numerical Domains

- Polyhedra Domain [Cousot & Halbwachs, POPL'78]
  - $a_1 x_1 + \ldots + a_n x_n \leq a_{n+1}$
  - $2x + y \leq 5$, but not $xy \leq 5$
  - Domain is infinite: $(x \leq y) \sqsubseteq (x \leq y + 1) \sqsubseteq (x \leq y + 2) \ldots$

A word on Numerical Domains

- Polyhedra Domain [Cousot & Halbwachs, POPL'78]
  - $a_1 x_1 + \ldots + a_n x_n \leq a_{n+1}$
  - $2x + y \leq 5$, but not $xy \leq 5$
  - Domain is infinite: $(x \leq y) \sqsubseteq (x \leq y + 1) \sqsubseteq (x \leq y + 2) \ldots$
  - To guarantee termination, widening is provided

A word on Numerical Domains

- Polyhedra Domain [Cousot & Halbwachs, POPL'78]
  - $a_1 x_1 + \ldots + a_n x_n \leq a_{n+1}$
  - $2x + y \leq 5$, but not $xy \leq 5$
  - Domain is infinite: $(x \leq y) \sqsubseteq (x \leq y + 1) \sqsubseteq (x \leq y + 2) \ldots$
  - To guarantee termination, widening is provided

- Octagon Domain [Mine, HOSC'06]

A word on Numerical Domains

- Polyhedra Domain [Cousot & Halbwachs, POPL'78]
  - $a_1 x_1 + \ldots + a_n x_n \leq a_{n+1}$
  - $2x + y \leq 5$, but not $xy \leq 5$
  - Domain is infinite: $(x \leq y) \sqsubseteq (x \leq y + 1) \sqsubseteq (x \leq y + 2) \ldots$
  - To guarantee termination, widening is provided

- Octagon Domain [Mine, HOSC'06]
  - $\pm x \pm y \leq c$

A word on Numerical Domains

- Polyhedra Domain [Cousot & Halbwachs, POPL'78]
  - $a_1 x_1 + \ldots + a_n x_n \leq a_{n+1}$
  - $2x + y \leq 5$, but not $xy \leq 5$
  - Domain is infinite: $(x \leq y) \sqsubseteq (x \leq y + 1) \sqsubseteq (x \leq y + 2) \ldots$
  - To guarantee termination, widening is provided

- Octagon Domain [Mine, HOSC'06]
  - $\pm x \pm y \leq c$
  - $x + y \leq 5$, $x - z \leq 8$ but not $2x + y \leq 5$

A word on Numerical Domains

- Polyhedra Domain [Cousot & Halbwachs, POPL'78]
  - $a_1 x_1 + \ldots + a_n x_n \leq a_{n+1}$
  - $2x + y \leq 5$, but not $xy \leq 5$
  - Domain is infinite: $(x \leq y) \sqsubseteq (x \leq y + 1) \sqsubseteq (x \leq y + 2) \ldots$
  - To guarantee termination, widening is provided

- Octagon Domain [Mine, HOSC'06]
  - $\pm x \pm y \leq c$
  - $x + y \leq 5$, $x - z \leq 8$ but not $2x + y \leq 5$
  - Domain infinite, widening is provided

Checking Conflict-Freedom

Step 1: Compute all reachable abstract states

Step 2: Check if the abstract states are conflict-free

Compute Abstract States

Compute invariants for each task type

Sequential analysis computes:
heap and numerical invariants per task type

Compute Abstract States: Example

```
void update (double[][] G, int start, int last, double c1, double c2, int nm, int ps) {
  finish {
    for (tid = start ; tid < last ; tid += 1) {
      async {
        int i = 2 * tid - ps;
        double[] Gi = G [i];
        double[] Gim = G [i - 1];
        double[] Gip = G [i + 1];
        for (int j=1; j<nm; j++) {
          double tmp1 = Gim[j]
          double tmp2 = Gip[j]
          double tmp3 = Gi[j-1]
          double tmp4 = Gi[j+1]
          double tmp5 = Gi[j];
          Gi[j] = c1 * (tmp1 + tmp2 + tmp3 + tmp4) + c2 * tmp5
    }}}}
```

Compute Abstract States: Example

```
void update (double[][] G, int start, int last, double c1, double c2, int nm, int ps) {
  finish {
    for (tid = start ; tid < last ; tid += 1) {
      async {
        int i = 2 * tid - ps;
 1:   double[] Gi = G [i];
 2:   double[] Gim = G [i - 1];
 3:   double[] Gip = G [i + 1];
       for (int j=1; j<nm; j++) {
 4:      double tmp1 = Gim[j]
 5:      double tmp2 = Gip[j]
 6:      double tmp3 = Gi[j-1]
 7:      double tmp4 = Gi[j+1]
 8:      double tmp5 = Gi[j];
 9:      Gi[j] = c1 * (tmp1 + tmp2 + tmp3 + tmp4) + c2 * tmp5
      } } } }
```

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Heap = (G $\rightarrow$ {A_G},
Gim $\rightarrow$ T,
Gip $\rightarrow$ T,
Gi $\rightarrow$ T)

Abstract States Concrete States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States Concrete States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1

$pc_1 = 1, ps = 0, idx_1 = 2$

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1

$pc_1 = 1, ps = 0, idx_1 = 2$

$pc_1 = 1, ps = 1, idx_1 = 1$

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1

$pc_1 = 1, ps = 0, idx_1 = 2$

$pc_1 = 1, ps = 1, idx_1 = 1$

$pc_1 = 1, ps = 2, idx_1 = 0$

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1 ..... tid = 4

$pc_1 = 1, ps = 0, idx_1 = 2$

$pc_1 = 1, ps = 1, idx_1 = 1$

$pc_1 = 1, ps = 2, idx_1 = 0$

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1 ..... tid = 4

$pc_1 = 1, ps = 0, idx_1 = 2$

$pc_1 = 1, ps = 1, idx_1 = 1$

$pc_1 = 1, ps = 2, idx_1 = 0$

$pc_4 = 1, ps = 0, idx_4 = 8$

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1	.....	tid = 4
$pc_1 = 1, ps = 0, idx_1 = 2$		$pc_4 = 1, ps = 0, idx_4 = 8$
$pc_1 = 1, ps = 1, idx_1 = 1$		$pc_4 = 1, ps = 3, idx_4 = 5$
$pc_1 = 1, ps = 2, idx_1 = 0$		

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1	.....	tid = 4
$pc_1 = 1, ps = 0, idx_1 = 2$		$pc_4 = 1, ps = 0, idx_4 = 8$
$pc_1 = 1, ps = 1, idx_1 = 1$		$pc_4 = 1, ps = 3, idx_4 = 5$
$pc_1 = 1, ps = 2, idx_1 = 0$		$pc_4 = 1, ps = 7, idx_4 = 1$
		.....

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1	.....	tid = 4	.....
$pc_1 = 1, ps = 0, idx_1 = 2$		$pc_4 = 1, ps = 0, idx_4 = 8$	
$pc_1 = 1, ps = 1, idx_1 = 1$		$pc_4 = 1, ps = 3, idx_4 = 5$	
$pc_1 = 1, ps = 2, idx_1 = 0$		$pc_4 = 1, ps = 7, idx_4 = 1$	
		.....	

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1	.....	tid = 4	.....
$pc_1 = 1, ps = 0, idx_1 = 2$		$pc_4 = 1, ps = 0, idx_4 = 8$	
$pc_1 = 1, ps = 1, idx_1 = 1$		$pc_4 = 1, ps = 3, idx_4 = 5$	
$pc_1 = 1, ps = 2, idx_1 = 0$		$pc_4 = 1, ps = 7, idx_4 = 1$	
		.....	

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1 ..... tid = 4 .....

$pc_1 = 1, ps = 0, idx_1 = 2$ | $pc_4 = 1, ps = 0, idx_4 = 8$

$pc_1 = 1, ps = 1, idx_1 = 1$ | $pc_4 = 1, ps = 3, idx_4 = 5$

$pc_1 = 1, ps = 2, idx_1 = 0$ | $pc_4 = 1, ps = 7, idx_4 = 1$

.....

tid = 1

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1 ..... tid = 4 .....

$pc_1 = 1, ps = 0, idx_1 = 2$ | $pc_4 = 1, ps = 0, idx_4 = 8$

$pc_1 = 1, ps = 1, idx_1 = 1$ | $pc_4 = 1, ps = 3, idx_4 = 5$

$pc_1 = 1, ps = 2, idx_1 = 0$ | $pc_4 = 1, ps = 7, idx_4 = 1$

.....

tid = 1

$pc_1 = 5, nm = 5, idx_1 = 4$

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1 ..... tid = 4 .....

$pc_1 = 1, ps = 0, idx_1 = 2$ | $pc_4 = 1, ps = 0, idx_4 = 8$

$pc_1 = 1, ps = 1, idx_1 = 1$ | $pc_4 = 1, ps = 3, idx_4 = 5$

$pc_1 = 1, ps = 2, idx_1 = 0$ | $pc_4 = 1, ps = 7, idx_4 = 1$

.....

tid = 1

$pc_1 = 5, nm = 5, idx_1 = 4$

$pc_1 = 5, nm = 9, idx_1 = 8$

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1 ..... tid = 4 .....

$pc_1 = 1, ps = 0, idx_1 = 2$ | $pc_4 = 1, ps = 0, idx_4 = 8$

$pc_1 = 1, ps = 1, idx_1 = 1$ | $pc_4 = 1, ps = 3, idx_4 = 5$

$pc_1 = 1, ps = 2, idx_1 = 0$ | $pc_4 = 1, ps = 7, idx_4 = 1$

.....

tid = 1

$pc_1 = 5, nm = 5, idx_1 = 4$

$pc_1 = 5, nm = 9, idx_1 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2$

.....

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States

tid = 1 ..... tid = 4 .....

$pc_1 = 1, ps = 0, idx_1 = 2$ | $pc_4 = 1, ps = 0, idx_4 = 8$

$pc_1 = 1, ps = 1, idx_1 = 1$ | $pc_4 = 1, ps = 3, idx_4 = 5$

$pc_1 = 1, ps = 2, idx_1 = 0$ | $pc_4 = 1, ps = 7, idx_4 = 1$

.....

tid = 1 ..... tid = 4

$pc_1 = 5, nm = 5, idx_1 = 4$

$pc_1 = 5, nm = 9, idx_1 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2$

.....

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

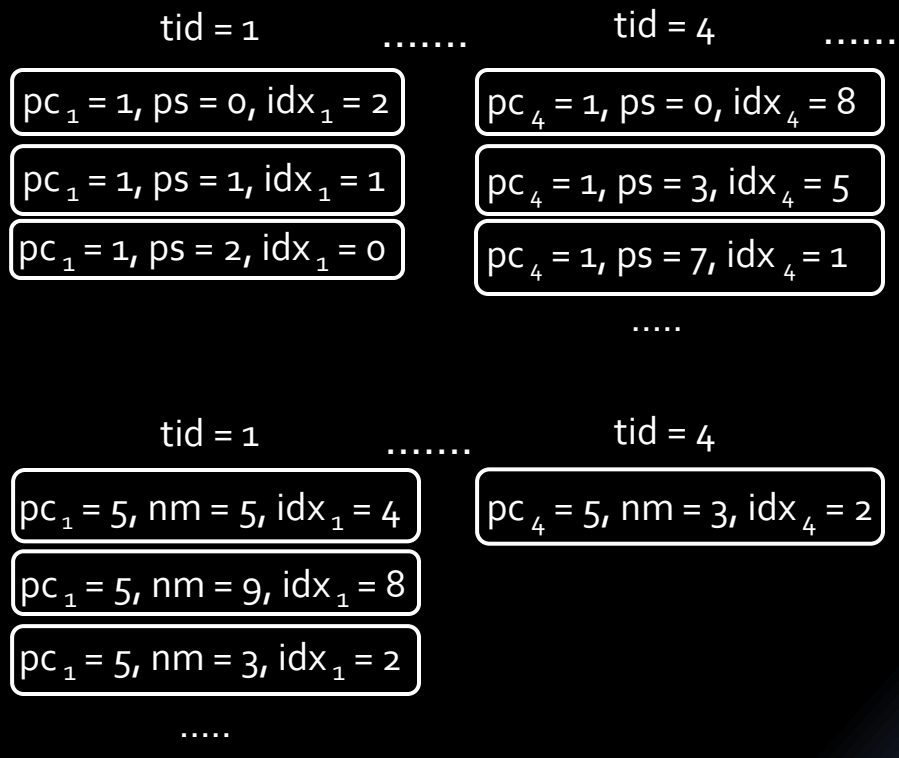
σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States



Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

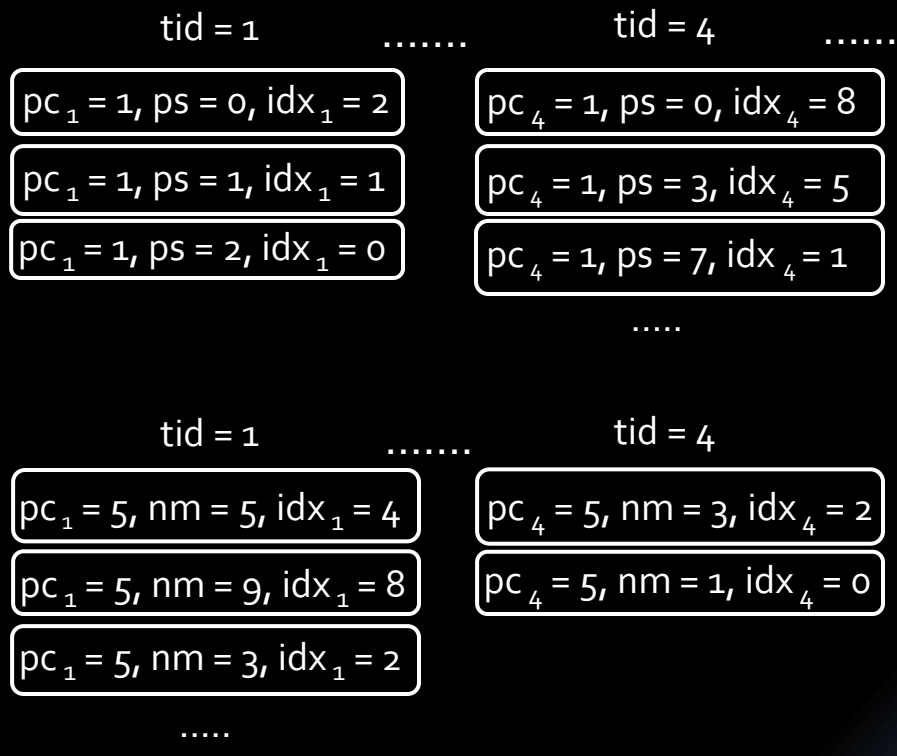
σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States



Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

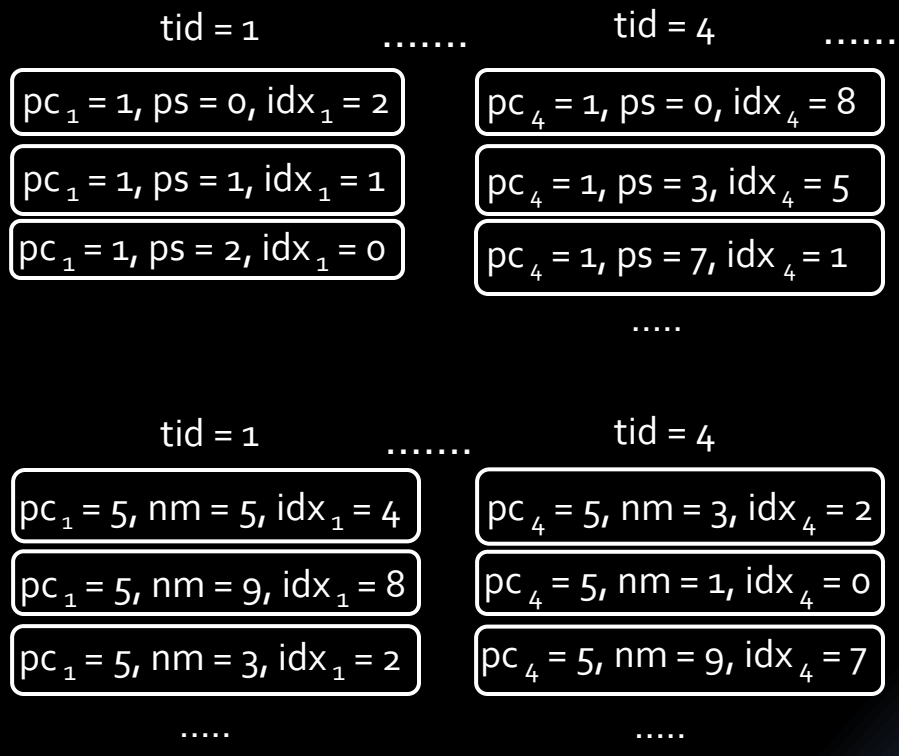
σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States



Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

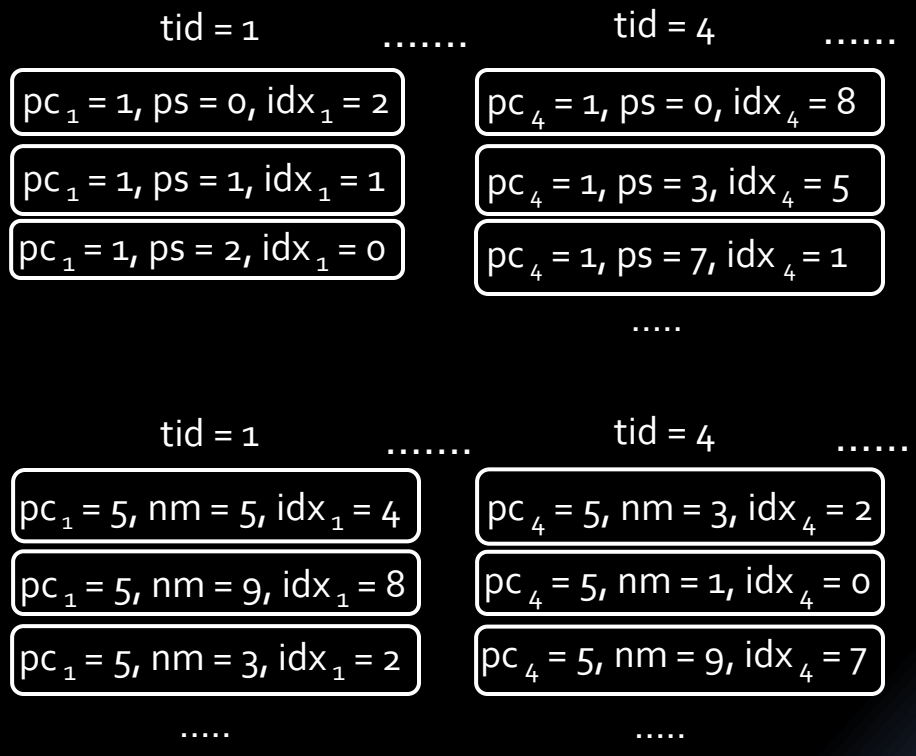
σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States



Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

Concrete States

tid = 1

$pc_1 = 1, ps = 0, idx_1 = 2$

$pc_1 = 1, ps = 1, idx_1 = 1$

$pc_1 = 1, ps = 2, idx_1 = 0$

tid = 4

$pc_4 = 1, ps = 0, idx_4 = 8$

$pc_4 = 1, ps = 3, idx_4 = 5$

$pc_4 = 1, ps = 7, idx_4 = 1$

tid = 1

$pc_1 = 5, nm = 5, idx_1 = 4$

$pc_1 = 5, nm = 9, idx_1 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2$

tid = 4

$pc_4 = 5, nm = 3, idx_4 = 2$

$pc_4 = 5, nm = 1, idx_4 = 0$

$pc_4 = 5, nm = 9, idx_4 = 7$

Program States

Concrete States

Program States

tid = 1

$pc_1 = 1, ps = 0, idx_1 = 2$

$pc_1 = 1, ps = 1, idx_1 = 1$

$pc_1 = 1, ps = 2, idx_1 = 0$

tid = 4

$pc_4 = 1, ps = 0, idx_4 = 8$

$pc_4 = 1, ps = 3, idx_4 = 5$

$pc_4 = 1, ps = 7, idx_4 = 1$

tid = 1

$pc_1 = 5, nm = 5, idx_1 = 4$

$pc_1 = 5, nm = 9, idx_1 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2$

tid = 4

$pc_4 = 5, nm = 3, idx_4 = 2$

$pc_4 = 5, nm = 1, idx_4 = 0$

$pc_4 = 5, nm = 9, idx_4 = 7$

Concrete States

tid = 1

$pc_1 = 1, ps = 0, idx_1 = 2$

$pc_1 = 1, ps = 1, idx_1 = 1$

$pc_1 = 1, ps = 2, idx_1 = 0$

tid = 4

$pc_4 = 1, ps = 0, idx_4 = 8$

$pc_4 = 1, ps = 3, idx_4 = 5$

$pc_4 = 1, ps = 7, idx_4 = 1$

tid = 1

$pc_1 = 5, nm = 5, idx_1 = 4$

$pc_1 = 5, nm = 9, idx_1 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2$

tid = 4

$pc_4 = 5, nm = 3, idx_4 = 2$

$pc_4 = 5, nm = 1, idx_4 = 0$

$pc_4 = 5, nm = 9, idx_4 = 7$

Program States

$pc_1 = 1, ps = 0, idx_1 = 2, pc_4 = 1, idx_4 = 8$

Concrete States

tid = 1

$pc_1 = 1, ps = 0, idx_1 = 2$

$pc_1 = 1, ps = 1, idx_1 = 1$

$pc_1 = 1, ps = 2, idx_1 = 0$

tid = 4

$pc_4 = 1, ps = 0, idx_4 = 8$

$pc_4 = 1, ps = 3, idx_4 = 5$

$pc_4 = 1, ps = 7, idx_4 = 1$

tid = 1

$pc_1 = 5, nm = 5, idx_1 = 4$

$pc_1 = 5, nm = 9, idx_1 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2$

tid = 4

$pc_4 = 5, nm = 3, idx_4 = 2$

$pc_4 = 5, nm = 1, idx_4 = 0$

$pc_4 = 5, nm = 9, idx_4 = 7$

Program States

$pc_1 = 1, ps = 0, idx_1 = 2, pc_4 = 1, idx_4 = 8$

Concrete States

tid = 1

$pc_1 = 1, ps = 0, idx_1 = 2$

$pc_1 = 1, ps = 1, idx_1 = 1$

$pc_1 = 1, ps = 2, idx_1 = 0$

tid = 4

$pc_4 = 1, ps = 0, idx_4 = 8$

$pc_4 = 1, ps = 3, idx_4 = 5$

$pc_4 = 1, ps = 7, idx_4 = 1$

tid = 1

$pc_1 = 5, nm = 5, idx_1 = 4$

$pc_1 = 5, nm = 9, idx_1 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2$

tid = 4

$pc_4 = 5, nm = 3, idx_4 = 2$

$pc_4 = 5, nm = 1, idx_4 = 0$

$pc_4 = 5, nm = 9, idx_4 = 7$

Program States

$pc_1 = 1, ps = 0, idx_1 = 2, pc_4 = 1, idx_4 = 8$

Concrete States

tid = 1

$pc_1 = 1, ps = 0, idx_1 = 2$

$pc_1 = 1, ps = 1, idx_1 = 1$

$pc_1 = 1, ps = 2, idx_1 = 0$

tid = 4

$pc_4 = 1, ps = 0, idx_4 = 8$

$pc_4 = 1, ps = 3, idx_4 = 5$

$pc_4 = 1, ps = 7, idx_4 = 1$

tid = 1

$pc_1 = 5, nm = 5, idx_1 = 4$

$pc_1 = 5, nm = 9, idx_1 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2$

tid = 4

$pc_4 = 5, nm = 3, idx_4 = 2$

$pc_4 = 5, nm = 1, idx_4 = 0$

$pc_4 = 5, nm = 9, idx_4 = 7$

Program States

$pc_1 = 1, ps = 0, idx_1 = 2, pc_4 = 1, idx_4 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2, pc_4 = 1, ps = 0, idx_4 = 2$

Concrete States

tid = 1

$pc_1 = 1, ps = 0, idx_1 = 2$

$pc_1 = 1, ps = 1, idx_1 = 1$

$pc_1 = 1, ps = 2, idx_1 = 0$

tid = 4

$pc_4 = 1, ps = 0, idx_4 = 8$

$pc_4 = 1, ps = 3, idx_4 = 5$

$pc_4 = 1, ps = 7, idx_4 = 1$

tid = 1

$pc_1 = 5, nm = 5, idx_1 = 4$

$pc_1 = 5, nm = 9, idx_1 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2$

tid = 4

$pc_4 = 5, nm = 3, idx_4 = 2$

$pc_4 = 5, nm = 1, idx_4 = 0$

$pc_4 = 5, nm = 9, idx_4 = 7$

Program States

$pc_1 = 1, ps = 0, idx_1 = 2, pc_4 = 1, idx_4 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2, pc_4 = 1, ps = 0, idx_4 = 2$

Concrete States

tid = 1

$pc_1 = 1, ps = 0, idx_1 = 2$

$pc_1 = 1, ps = 1, idx_1 = 1$

$pc_1 = 1, ps = 2, idx_1 = 0$

tid = 4

$pc_4 = 1, ps = 0, idx_4 = 8$

$pc_4 = 1, ps = 3, idx_4 = 5$

$pc_4 = 1, ps = 7, idx_4 = 1$

tid = 1

$pc_1 = 5, nm = 5, idx_1 = 4$

$pc_1 = 5, nm = 9, idx_1 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2$

tid = 4

$pc_4 = 5, nm = 3, idx_4 = 2$

$pc_4 = 5, nm = 1, idx_4 = 0$

$pc_4 = 5, nm = 9, idx_4 = 7$

Program States

$pc_1 = 1, ps = 0, idx_1 = 2, pc_4 = 1, idx_4 = 8$

$pc_1 = 5, nm = 3, idx_1 = 2, pc_4 = 1, ps = 0, idx_4 = 2$

$pc_1 = 5, nm = 3, idx_1 = 2, pc_4 = 5, idx_4 = 2$

Abstract States

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

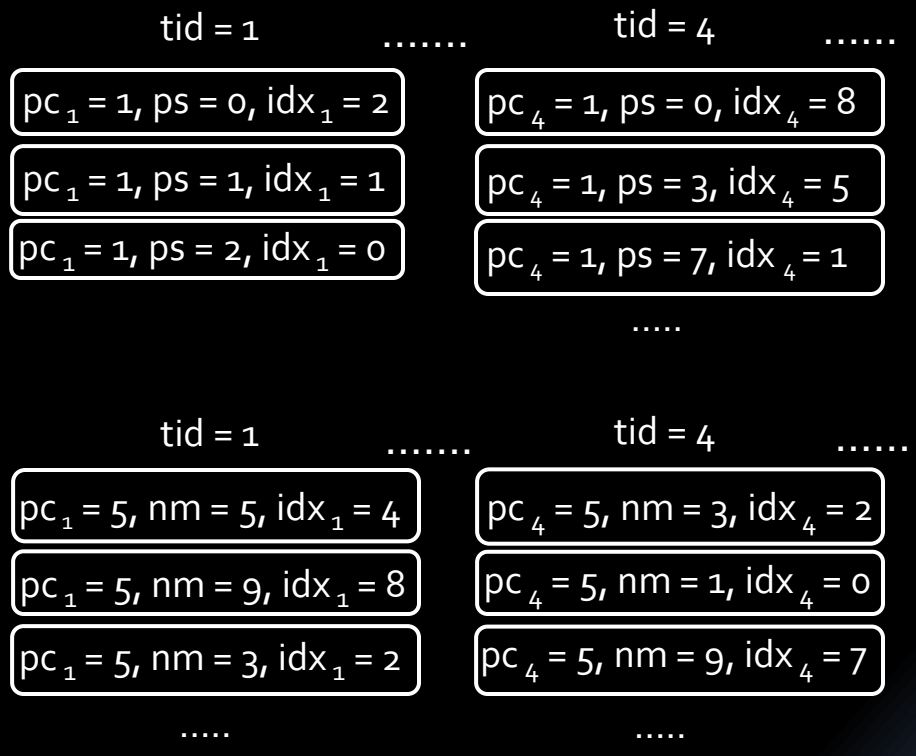
σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Concrete States



Heap = (G $\rightarrow$ {A_G},
Gim $\rightarrow$ T,
Gip $\rightarrow$ T,
Gi $\rightarrow$ T)

Abstract Conflict-Free Checker

For every pair of abstract states σ_i , σ_k:

if st_i in σ_i and st_k in σ_k may happen in parallel

A = combine σ_i , σ_k and ($tid_i \neq tid_k$)

if ALoc(st_i) and ALoc(st_k) may equal in A

exit ("Program may be non-deterministic")

exit ("Program is deterministic")

Abstract Conflict-Free Checker

For every pair of abstract states σ_i , σ_k:

Temporal check

if st_i in σ_i and st_k in σ_k may happen in parallel

A = combine σ_i , σ_k and ($tid_i \neq tid_k$)

if ALoc(st_i) and ALoc(st_k) may equal in A

exit ("Program may be non-deterministic")

exit ("Program is deterministic")

Abstract Conflict-Free Checker

For every pair of abstract states σ_i , σ_k:

Temporal check

if st_i in σ_i and st_k in σ_k may happen in parallel

A = combine σ_i , σ_k and ($tid_i \neq tid_k$)

if ALoc(st_i) and ALoc(st_k) may equal in A

exit ("Program may be non-deterministic")

Spatial check

exit ("Program is deterministic")

May happen in parallel (Temporal Check)

For every pair of abstract states σ_i , σ_k:

if st_i in σ_l and st_k in σ_k may happen in parallel

A = combine σ_i, σ_k and ($tid_i \neq tid_k$)

if ALoc(st_i) and ALoc(st_k) may equal in A

exit ("Program may be non-deterministic")

exit ("Program is deterministic")

May happen in parallel (Temporal Check)

For every pair of abstract states σ_i , σ_k:

if st_i in σ_l and st_k in σ_k may happen in parallel

A = combine σ_i, σ_k ($tid_i \neq tid_k$)

if A

- Can be answered with existing may-happen analyses
- Cheaper for structured languages
- Simple for many structured applications

exit ("Program is deterministic")

May Equal (Spatial Check)

For every pair of abstract states σ_i , σ_k:

if st_i in σ_i and st_k in σ_k may happen in parallel

A = combine σ_l, σ_k and ($tid_i \neq tid_k$)

if ALoc(st_i) and ALoc(st_k) may equal in A

exit ("Program may be non-deterministic")

exit ("Program is deterministic")

May Equal (Spatial Check)

ALoc(st) := (base variable, index variable)

Example: ALoc (G[i] = 128) := (G, i)

May Equal (Spatial Check)

ALoc(st) := (base variable, index variable)

Example: ALoc (G[i] = 128) := (G, i)

May ALoc(st_1) := (a, i) equal ALoc(st_2) := (b, k) in abstract state A ?

May Equal (Spatial Check)

ALoc(st) := (base variable, index variable)

Example: ALoc (G[i] = 128) := (G, i)

May ALoc(st_1) := (a, i) equal ALoc(st_2) := (b, k) in abstract state A ?

if (a may alias b) and (i may overlap k)

Yes

May Equal (Spatial Check)

ALoc(st) := (base variable, index variable)

Example: ALoc (G[i] = 128) := (G, i)

May ALoc(st_1) := (a, i) equal ALoc(st_2) := (b, k) in abstract state A ?

if (a may alias b) and (i may overlap k)

Yes

else

No

May Equal (Spatial Check)

ALoc(st) := (base variable, index variable)

Example: ALoc (G[i] = 128) := (G, i)

May ALoc(st_1) := (a, i) equal ALoc(st_2) := (b, k) in abstract state A ?

if (a may alias b) and (i may overlap k)

Yes

else

No

points-to (a) ∩ points-to(b) ≠ {}

May Equal (Spatial Check)

ALoc(st) := (base variable, index variable)

Example: ALoc (G[i] = 128) := (G, i)

May ALoc(st_1) := (a, i) equal ALoc(st_2) := (b, k) in abstract state A ?

if (a may alias b) and (i may overlap k)

Yes

else

No

A ⊓ (i = k) ≠ ⊥

points-to (a) ∩ points-to(b) ≠ {}

Compute Abstract States: Example

```
void update (double[][] G, int start, int last, double c1, double c2, int nm, int ps) {
  finish {
    for (tid = start ; tid < last ; tid += 1) {
      async {
        int i = 2 * tid - ps;
 1:   double[] Gi = G [i];
 2:   double[] Gim = G [i - 1];
 3:   double[] Gip = G [i + 1];
       for (int j=1; j<nm; j++) {
 4:      double tmp1 = Gim[j]
 5:      double tmp2 = Gip[j]
 6:      double tmp3 = Gi[j-1]
 7:      double tmp4 = Gi[j+1]
 8:      double tmp5 = Gi[j];
 9:      Gi[j] = c1 * (tmp1 + tmp2 + tmp3 + tmp4) + c2 * tmp5
      } } } }
```

Heap = (G → {A_G},
Gim → T,
Gip → T,
Gi → T)

σ_1 = {pc = 1, Heap, idx = 2*tid - ps}

σ_2 = {pc = 2, Heap, idx = 2*tid - ps – 1}

σ_3 = {pc = 3, Heap, idx = 2*tid - ps + 1}

σ_4 = {pc = 4, Heap, 1 ≤ idx < nm}

σ_5 = {pc = 5, Heap, 1 ≤ idx < nm}

σ_6 = {pc = 6, Heap, 0 ≤ idx < nm-1}

σ_7 = {pc = 7, Heap, 2 ≤ idx < nm+1}

σ_8 = {pc = 8, Heap, 1 ≤ idx < nm}

σ_9 = {pc = 9, Heap, 1 ≤ idx < nm}

Pick a pair of abstract states

σ_6 = ($pc_1 = 6$, ($G_1 \rightarrow$ {A_G}, $Gim_1 \rightarrow$ T, $Gip_1 \rightarrow$ T, $Gi_1 \rightarrow$ T), $0 \leq idx_1 < nm\text{-}1$)

σ_9 = ($pc_2 = 9$, ($G_2 \rightarrow$ {A_G}, $Gim_2 \rightarrow$ T, $Gip_2 \rightarrow$ T, $Gi_2 \rightarrow$ T), $1 \leq idx_2 < nm$)

May happen in parallel (Temporal Check)

$\sigma_6 = (pc_1 = 6, (G_1 \rightarrow \{A\_G\}, Gim_1 \rightarrow T, Gip_1 \rightarrow T, Gi_1 \rightarrow T), 0 \leq idx_1 < nm\text{-}1)$

$\sigma_9 = (pc_2 = 9, (G_2 \rightarrow \{A\_G\}, Gim_2 \rightarrow T, Gip_2 \rightarrow T, Gi_2 \rightarrow T), 1 \leq idx_2 < nm)$

Statements:

6: $tmp3 = Gi_1[idx_1]$ vs. 9: $Gi_2[idx_2] = \ldots$

May happen in parallel (Temporal Check)

σ_6 = ($pc_1 = 6$, ($G_1 \rightarrow \{A\_G\}$, $Gim_1 \rightarrow T$, $Gip_1 \rightarrow T$, $Gi_1 \rightarrow T$), $0 \leq idx_1 < nm\text{-}1$)

σ_9 = ($pc_2 = 9$, ($G_2 \rightarrow \{A\_G\}$, $Gim_2 \rightarrow T$, $Gip_2 \rightarrow T$, $Gi_2 \rightarrow T$), $1 \leq idx_2 < nm$)

Statements:

6: $tmp3 = Gi_1[idx_1]$ vs. 9: $Gi_2[idx_2] = \ldots$

May Happen in Parallel ? Yes

May Equal (Spatial Check)

σ_6 = ($pc_1 = 6$, ($G_1 \rightarrow$ {A_G}, $Gim_1 \rightarrow T$, $Gip_1 \rightarrow T$, $Gi_1 \rightarrow T$), $0 \leq idx_1 < nm-1$)

σ_9 = ($pc_2 = 9$, ($G_2 \rightarrow$ {A_G}, $Gim_2 \rightarrow T$, $Gip_2 \rightarrow T$, $Gi_2 \rightarrow T$), $1 \leq idx_2 < nm$)

Statements:

6: $tmp3 = Gi_1[idx_1]$ vs. 9: $Gi_2[idx_2] = \ldots$

Abstract Locations:

$ALoc_6 := (Gi_1, idx_1)$ vs. $ALoc_9 := (Gi_2, idx_2)$

May Equal (Spatial Check)

σ_6 = ($pc_1 = 6$, ($G_1 \rightarrow \{A\_G\}$, $Gim_1 \rightarrow T$, $Gip_1 \rightarrow T$, $Gi_1 \rightarrow T$), $0 \leq idx_1 < nm\text{-}1$)

σ_9 = ($pc_2 = 9$, ($G_2 \rightarrow \{A\_G\}$, $Gim_2 \rightarrow T$, $Gip_2 \rightarrow T$, $Gi_2 \rightarrow T$), $1 \leq idx_2 < nm$)

Statements:

6: $tmp3 = Gi_1[idx_1]$ vs. 9: $Gi_2[idx_2] = \ldots$

Abstract Locations:

$ALoc_6 := (Gi_1, idx_1)$ vs. $ALoc_9 := (Gi_2, idx_2)$

may Gi_1 alias Gi_2 ? Yes

may idx_1 overlap idx_2 ? Yes

May Equal (Spatial Check)

$\sigma_6 = (pc_1 = 6, (G_1 \rightarrow \{A\_G\}, Gim_1 \rightarrow T, Gip_1 \rightarrow T, Gi_1 \rightarrow T), 0 \leq idx_1 < nm\text{-}1)$

$\sigma_9 = (pc_2 = 9, (G_2 \rightarrow \{A\_G\}, Gim_2 \rightarrow T, Gip_2 \rightarrow T, Gi_2 \rightarrow T), 1 \leq idx_2 < nm)$

Statements:

6: $tmp3 = Gi_1[idx_1]$ vs. 9: $Gi_2[idx_2] = \ldots$

Program may be non-deterministic

Abstract Locations:

$ALoc_6 := (Gi_1, idx_1)$ vs. $ALoc_9 := (Gi_2, idx_2)$

may Gi_1 alias Gi_2 ? Yes

may idx_1 overlap idx_2 ? Yes

Heap abstraction imprecise

Aliasing information inside reference arrays lost

Example: Object A[], double G[][]

Points-to information not enough

Heap abstraction imprecise

Property Wanted: references inside array are distinct

$$\forall\ i.j.A.\ i \neq j \Rightarrow A[i] \neq A[j]$$

Very difficult to prove in general !

Simple specific solution

- In most numerical HPC programs
  - Reference arrays are initialized only with fresh objects
    - Example: A[i] = new Object();
  - Never updated after (parallel tasks do not modify A)
  - Holds in all of our benchmarks

- Easy to prove as a global invariant
  - Initialization in a single procedure
  - Simple analysis

Spatial check revisited: may equal ?

σ_6 = ($pc_1 = 6$, ($G_1 \rightarrow$ {A_G}, $Gim_1 \rightarrow T$, $Gip_1 \rightarrow T$, $Gi_1 \rightarrow T$), $0 \leq idx_1 < nm\text{-}1$)

σ_9 = ($pc_2 = 9$, ($G_2 \rightarrow$ {A_G}, $Gim_2 \rightarrow T$, $Gip_2 \rightarrow T$, $Gi_2 \rightarrow T$), $1 \leq idx_2 < nm$)

Statements:

6: $tmp3 = Gi_1[idx_1]$ vs. 9: $Gi_2[idx_2] = \ldots$

Spatial check revisited: may equal ?

σ_6 = ($pc_1 = 6$, ($G_1 \rightarrow$ {A_G}, $Gim_1 \rightarrow T$, $Gip_1 \rightarrow T$, $Gi_1 \rightarrow T$), $0 \leq idx_1 < nm\text{-}1$)

σ_9 = ($pc_2 = 9$, ($G_2 \rightarrow$ {A_G}, $Gim_2 \rightarrow T$, $Gip_2 \rightarrow T$, $Gi_2 \rightarrow T$), $1 \leq idx_2 < nm$)

Statements:

6: $tmp3 = Gi_1[idx_1]$ vs. 9: $Gi_2[idx_2] = \ldots$

Abstract Locations:

$ALoc_6 := ({}_Gi1, idx1)$ vs. $ALoc_9 := ({}_Gi2, idx2)$

Spatial check revisited: may equal ?

$\sigma_6 = (pc_1 = 6, (G_1 \rightarrow \{A\_G\}, Gim_1 \rightarrow T, Gip_1 \rightarrow T, Gi_1 \rightarrow T), 0 \leq idx_1 < nm\text{-}1)$

$\sigma_9 = (pc_2 = 9, (G_2 \rightarrow \{A\_G\}, Gim_2 \rightarrow T, Gip_2 \rightarrow T, Gi_2 \rightarrow T), 1 \leq idx_2 < nm)$

Statements:

6: $tmp3 = Gi_1[idx_1]$ vs. 9: $Gi_2[idx_2] = \ldots$

Abstract Locations:

$ALoc_6 := ({}_G i1, idx1)$ vs. $ALoc_9 := ({}_G i2, idx2)$

may Gi_1 alias Gi_2 ? No

(using $\forall$ i,j . $i \neq j \Rightarrow A\_G[i] \neq A\_G[j]$)

Spatial check revisited: may equal ?

σ_6 = ($pc_1 = 6$, ($G_1 \rightarrow$ {A_G}, $Gim_1 \rightarrow$ T, $Gip_1 \rightarrow$ T, $Gi_1 \rightarrow$ T), $0 \le idx_1 < nm-1$)

σ_9 = ($pc_2 = 9$, ($G_2 \rightarrow$ {A_G}, $Gim_2 \rightarrow$ T, $Gip_2 \rightarrow$ T, $Gi_2 \rightarrow$ T), $1 \le idx_2 < nm$)

Statements:

6: $tmp3 = Gi_1[idx_1]$ vs. 9: $Gi_2[idx_2] = \ldots$

Program is deterministic

$ALoc_6 := (_G i1, idx1)$ vs. $ALoc_9 := (_G i2, idx2)$

may Gi_1 alias Gi_2 ? No

(using $\forall$ i,j . $i \neq j \Rightarrow A\_G[i] \neq A\_G[j]$)

Outline

- Motivation
- Checking: concrete (finite) reasoning
- Proving: abstract (infinite) reasoning
- Experimental results
- Limitations

Implementation

- Soot
  - Analysis works on Jimple
  - Can generate a lot of variables
  - Need to identify loops (to know where to widen)
- Apron library
  - For computing numerical invariants
- Benchmarks
  - Used adapted Java JGF benchmarks

Results (Numerical analysis)

Program	LOC	Vars	Domain	Time (s)	Widen	Result
Crypt	110	180	Polyhedra	54	No	✓
Sor	35	21	Polyhedra	0.41	Yes	✓
Lufact	32	22	Octagon	1.94	Yes	✓
Series	67	14	Octagon	55	No	✓
Moldyn	340	175	Polyhedra, Octagon	28.8	Yes, No	✓
Sparse						✗
Raytracer						✗
Montecarlo						✗

Limitations

- Intra-Procedural
  - Had to inline some functions
  - We failed to inline in one benchmark
- Cannot handle pseudo non-linear constraints
  - A[x*N + z] = c, where N never changes
- Atomic sections
  - atomic {x = x + 1; }
- Accesses from nested primitive arrays
  - A[B[i]] = 5, where B[i]'s are distinct

Related Work

- Static analysis
  - Rugina & Rinard (ACM TOPLAS'2005)
    - Takes a different approach: assume no widening
    - Do not handle reference arrays
  - Race detection
- Dynamic analysis
  - Feng & Leiserson (SPAA 1997), Sadowski et. al (ESOP2009), Burnim & Sen (FSE 2009), Raman et. al (RV 2010)
- Programming Models & Language Runtimes (Dynamic)
  - Determinator (Yale), DMP (Washington), Grace (Umass) Kendo (MIT), Revisions (Microsoft)

Conclusion

- Automatic Verification of Determinism
  - verify stronger property: conflict-freedom
  - sequential analysis based on numerical domains
- Analysis implemented in Soot
  - consumes Java
  - uses Apron for numerical domains
- Future Work
  - handle certain linear constraints, interference, better array content analysis

Thanks for your attention ☺