What happens between communications?

Towards analysis of synchronization and parallelism in multi-threaded programs

Nina Bohr
John Gallagher
Morten Rhiger
Mads Rosendahl
Roskilde University

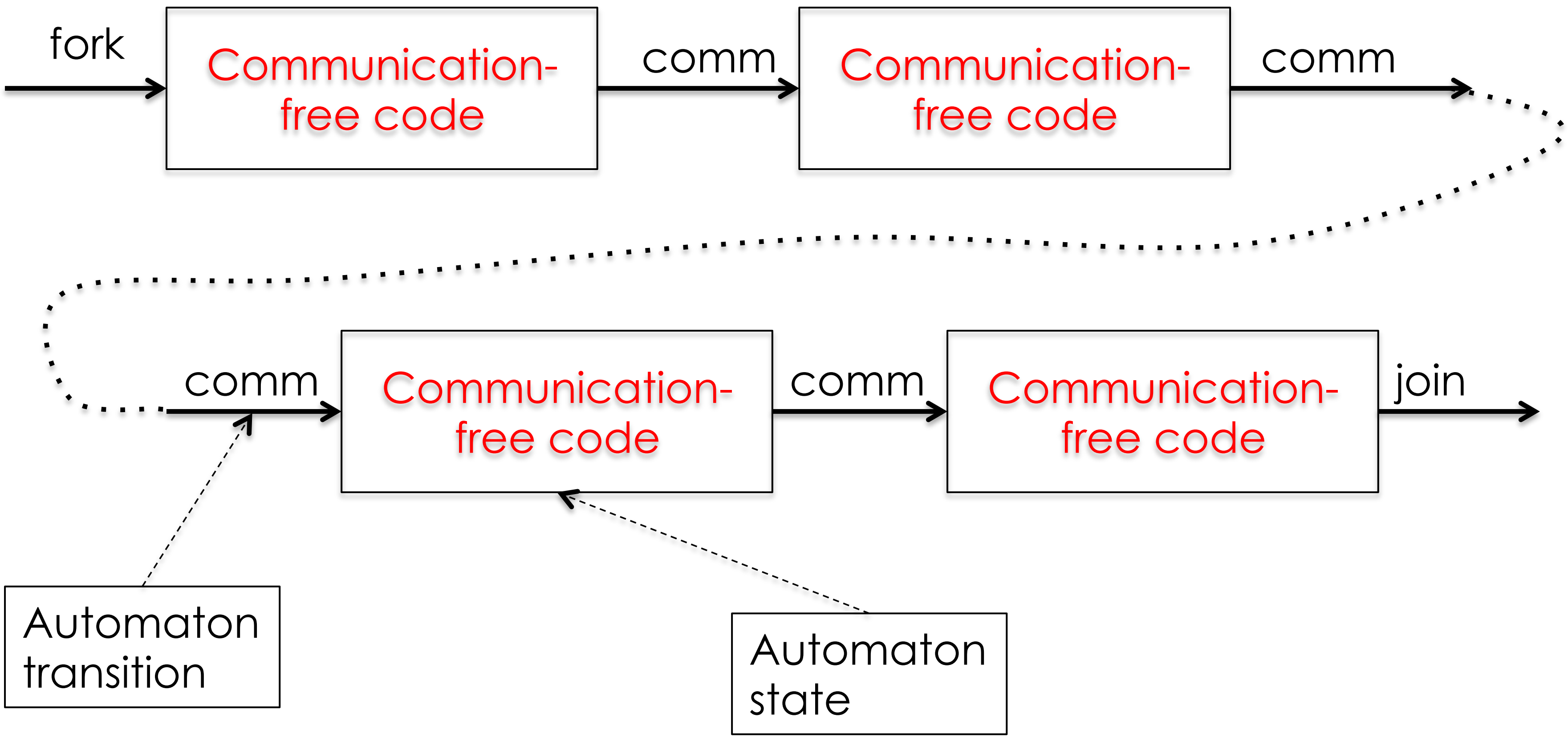
Some inter-related questions relevant to resource analysis of multi-threaded code (1)

- What resources are used by each thread between communications?
- What statements in different threads may run in parallel with each other?
  - → equivalently – which statements in different threads definitely do not run in parallel?

Some inter-related questions relevant to resource analysis of multi-threaded code (2)

- How (in)active is each thread? Are thread loads reasonably balanced?
- Are threads located on the appropriate cores?

Behaviour of a single thread

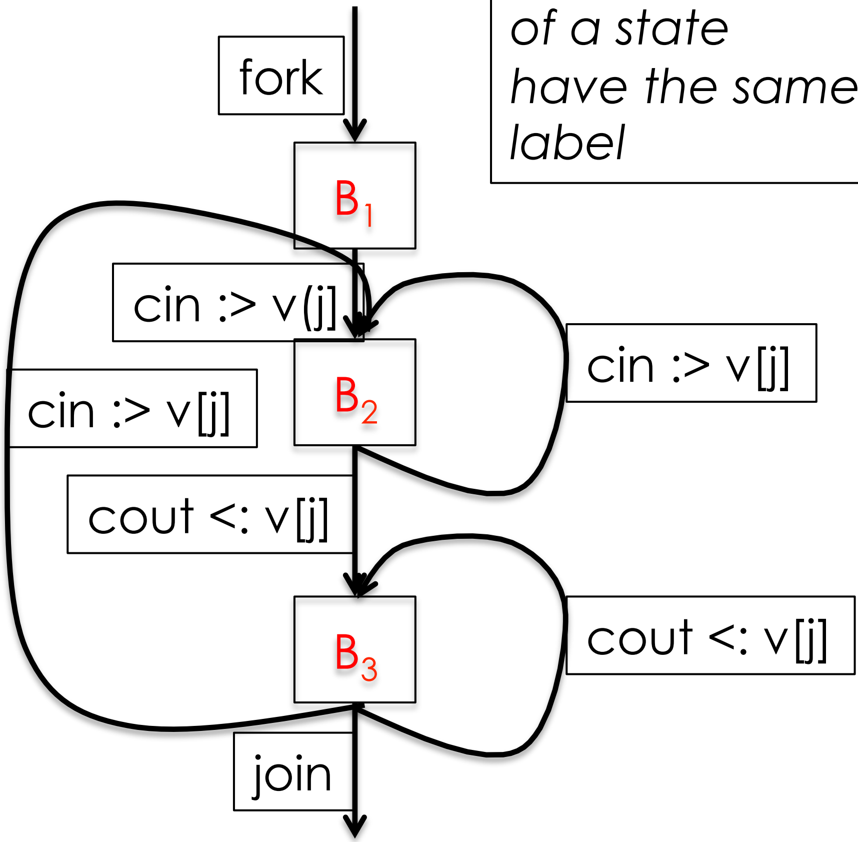


Overview of approach

- Construct an automaton for each thread
  - states are pieces of code that do not communicate
  - transitions are labelled with events
    - get, send, get+send, fork, join
- The automaton for a program is formed as a synchronised product of the automata for individual threads
  - synchronous communication (XC)

Automaton for a thread

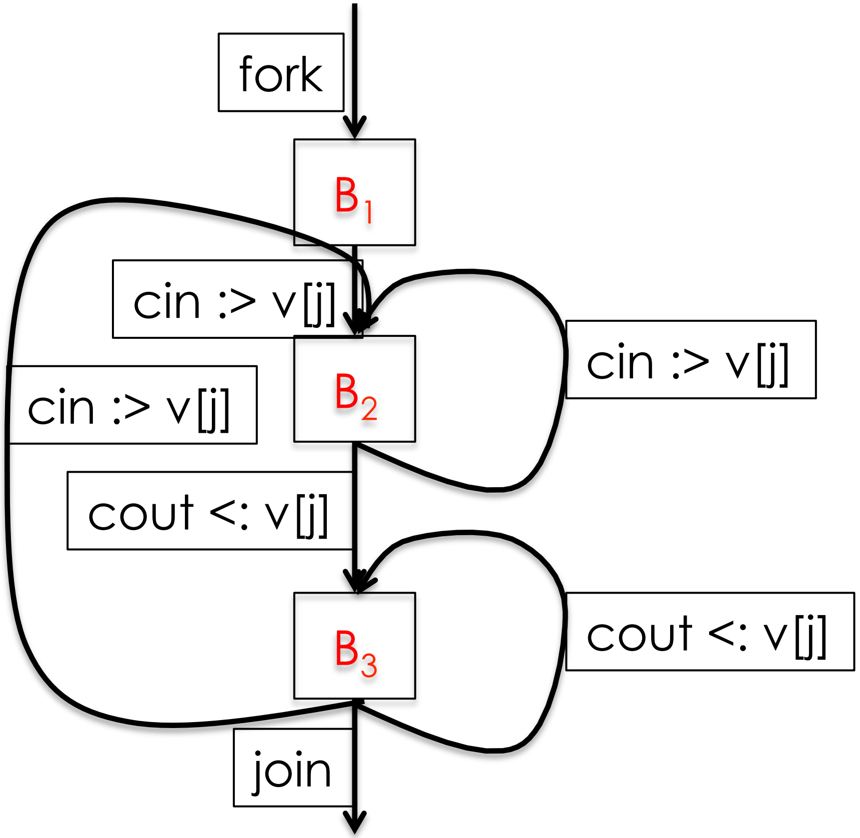
```
// assume N>0, M>0
// B1 starts here
while (1) {
    int v;
    for (int i = 0; i < N; i = i+ M) {

        for (int j = 0; j < M; j = j + 1) {
            cin :> v[j]; // B2 starts here
        }

        ......
        for (int j = 0; j < M; j = j + 1) {
            cout <: v[j]; // B3 starts here
        }
    }
}
```



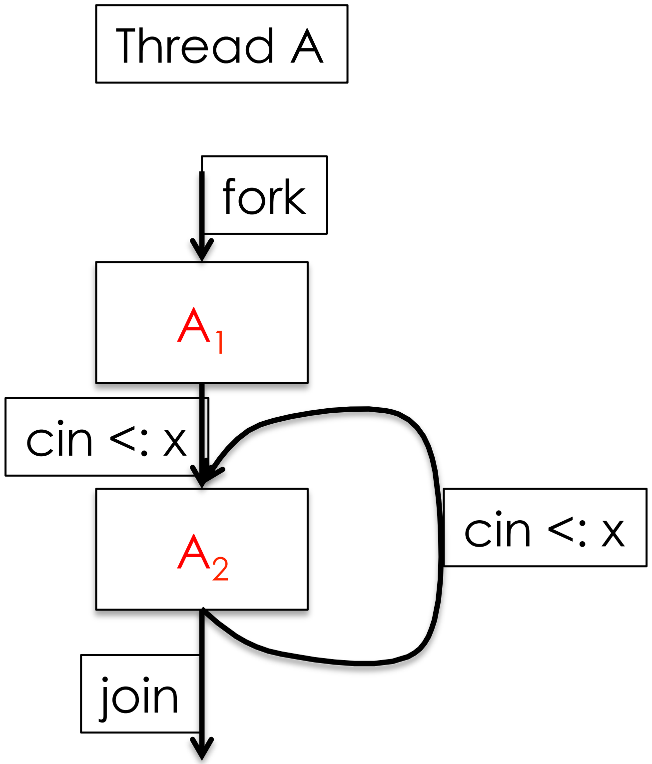
Note: all in-arcs of a state have the same label

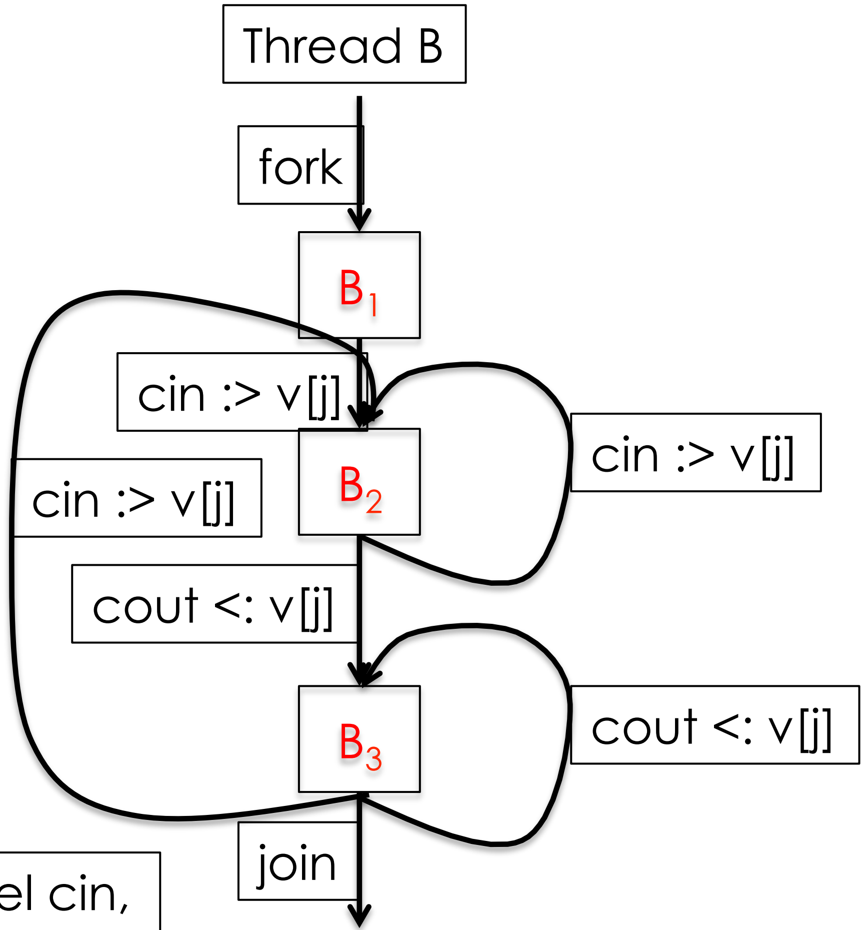
Notes on the automaton

1. Program statements can be in more than one state.
   – e.g. B_1 and B_2 overlap
2. There can be multiple paths through a state (possibly an infinite number)
   – e.g. B_3 can (1) re-enter inner for-loop, (2) re-enter outer for-loop, (3) exit both for-loops
3. Size of automaton is linear in code size.



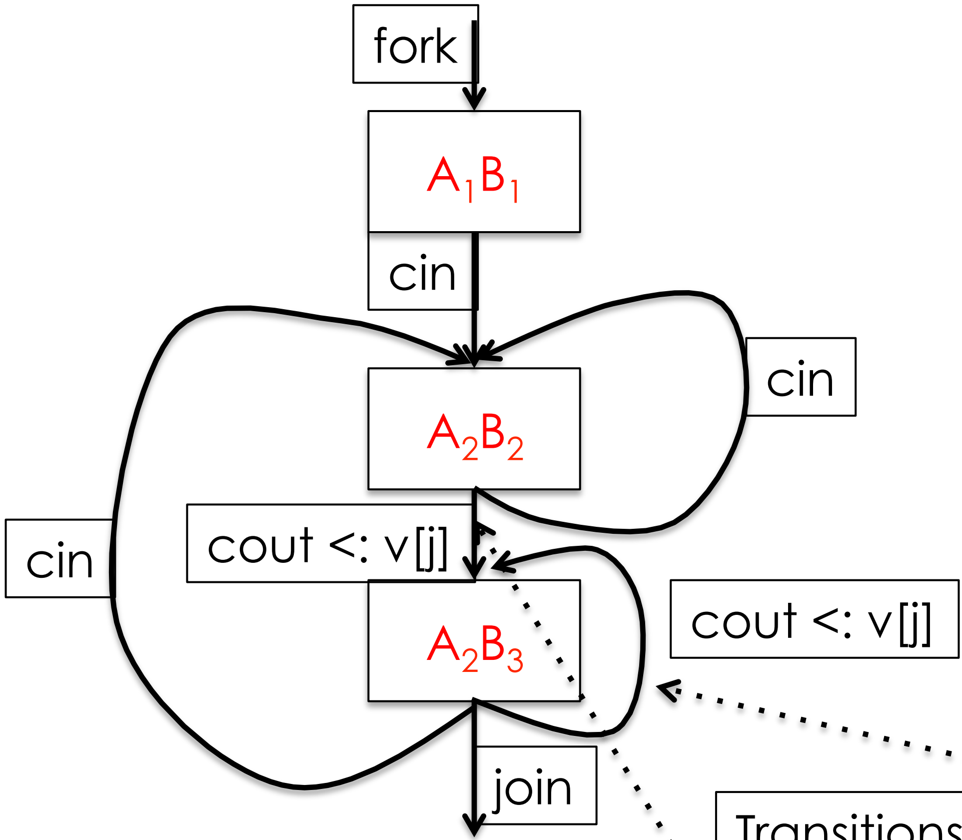
Parallel threads





Threads A and B synchronize on channel cin, but cout is not shared between the threads.

Product automaton



Product synchronizes internally on cin.

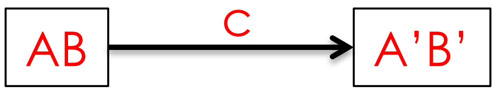
Still has unsynchronized transitions on cout.

In XC, possible deadlock is detected if the shared channel transitions do not match in the product.

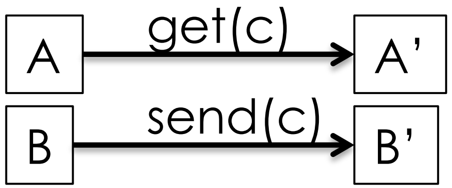
Transitions on the non-shared channel cout. Thread A remains in state A_2 while thread B can move to a different state.

Formal definition of product

AB is reachable if AB is the initial state, or



if AB reachable, c is a shared channel, where



and vice versa

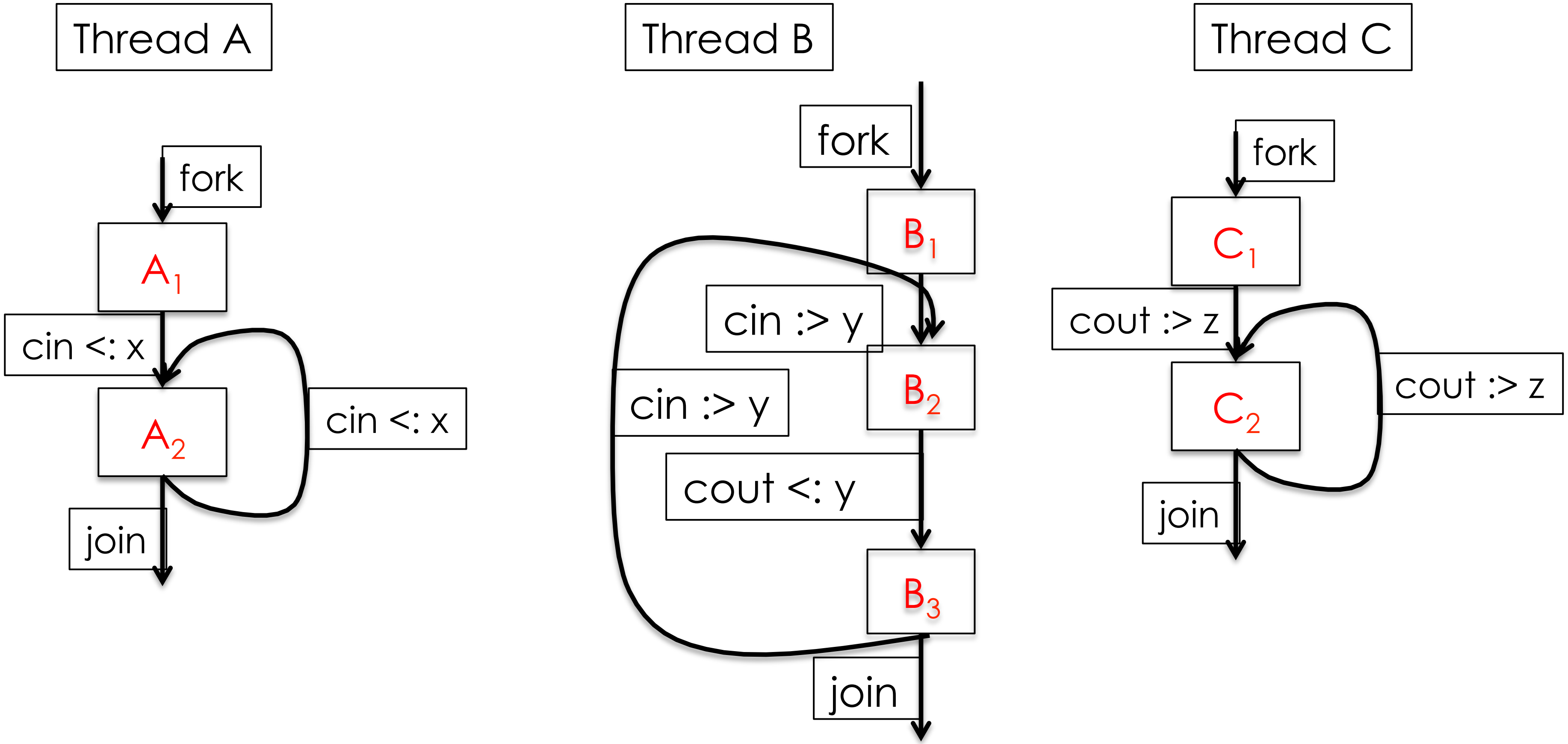
if AB reachable, c is not a shared channel, where

similarly for send(c), and symmetric cases

May-run-in-parallel analysis

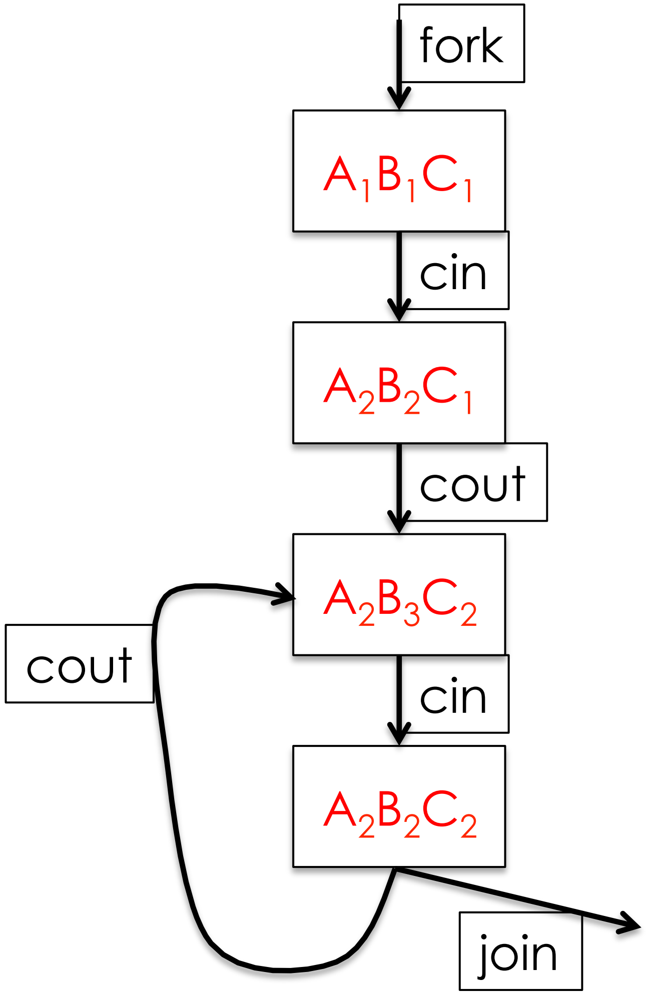
- Pair-states that are reachable in the product contain code that may run in parallel
  - i.e. A_1B_1, A_2B_2 and A_2B_3
  - pairs that do not occur cannot execute in parallel
- Analysing execution times of the elements of the pairs yields information on whether one thread may wait for the other.
  - could be WCET (if bounded) or parametric complexity analysis

Example - pipeline

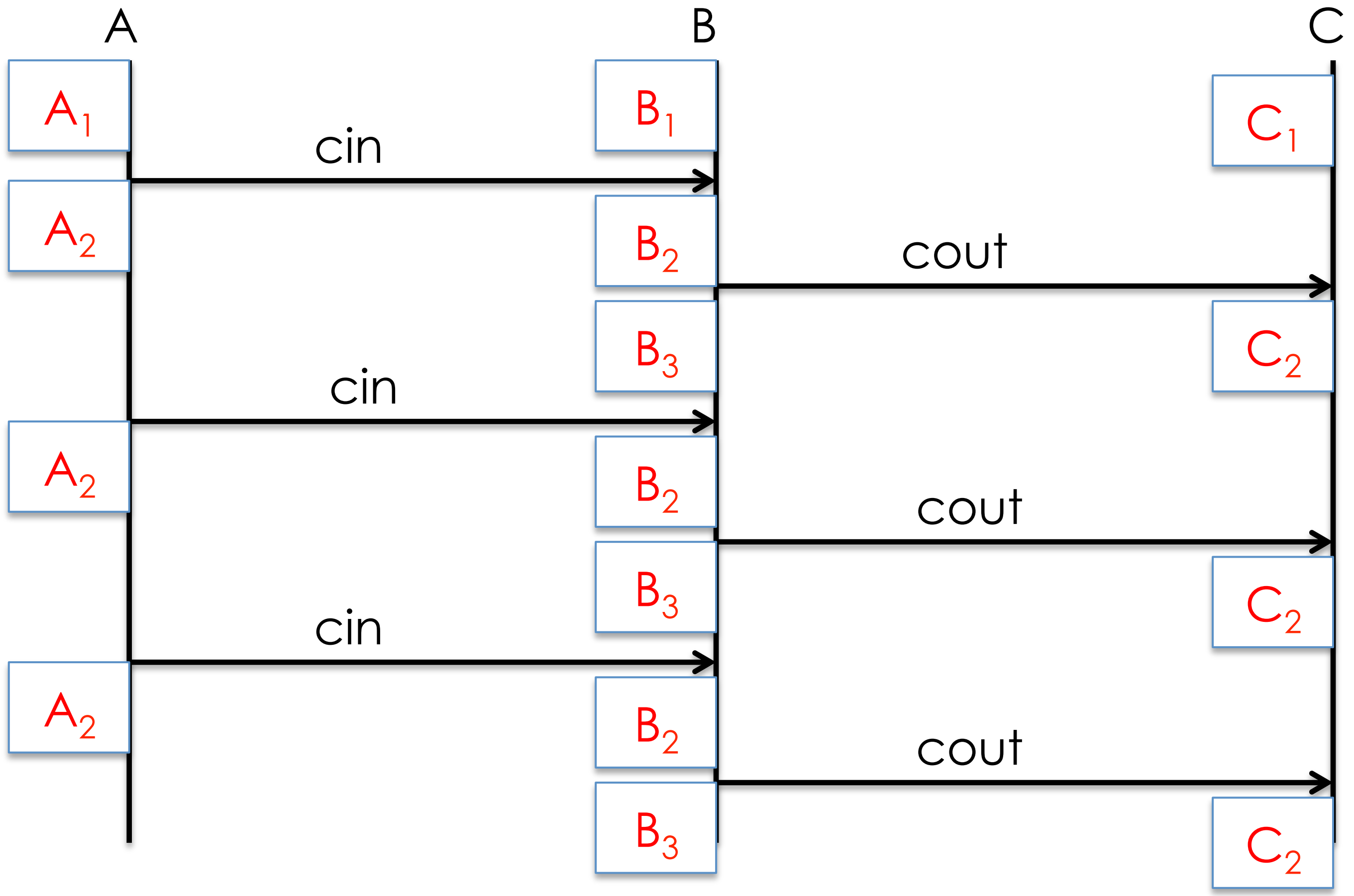


ENTRA

Pipeline product



Pipeline sequence diagram



Pipeline product states

- Reachable states (out of 12 possible)
  - $A_1B_1C_1$
  - $A_2B_2C_1$
  - $A_2B_3C_2$
  - $A_2B_2C_2$
- Timing analysis might show that A and C are hardly ever active simultaneously
  - should be allocated on the same core?
  - or A and C should run on slower cores?

Abstracting paths/states

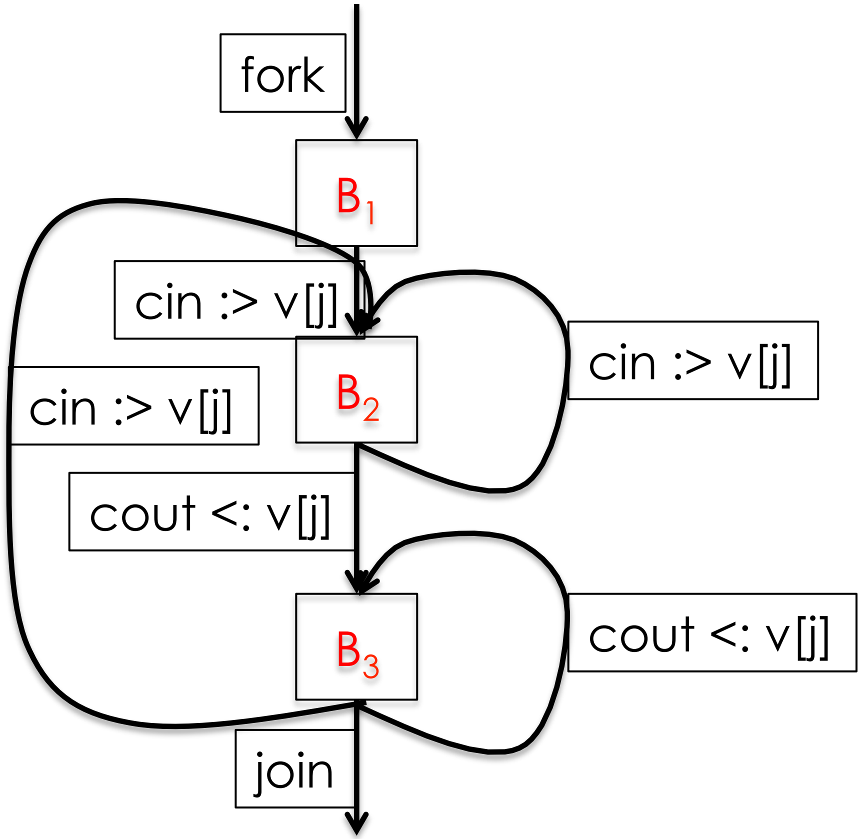
- Let the variables of a state be **X**.
- The behaviour of each state S can be abstracted to a relation $R_S(\mathbf{X},\mathbf{X'})$
  - e.g. abstract interpretation of the code.
  - relation between the values of the variables before (**X**) and after (**X'**) execution of the state.

Reachability relation

- State S' is reachable with state X' if
  - there is a transition from S to S', and
  - state S is reachable with state X, and
  - $R_S(X,X')$ holds
- Representation as Horn clause

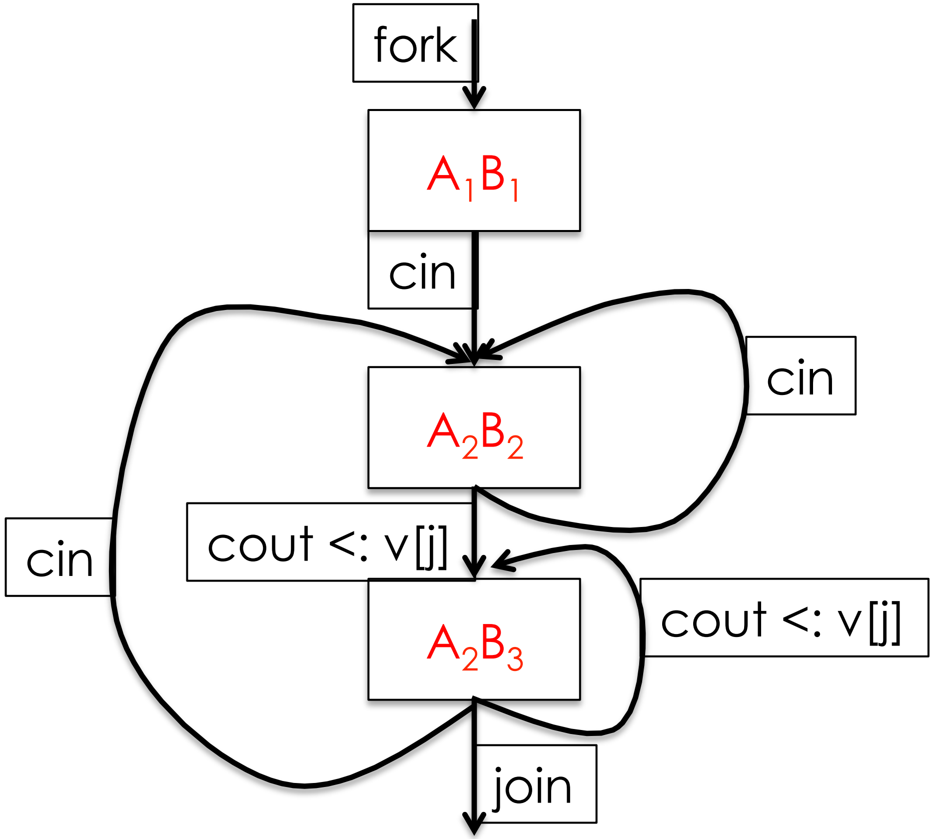
$reach_{S'}(X') \text{ :- } reach_S(X), R_S(X,X').$

Horn clause abstraction of automaton




```
reach1(X) :- init(X).

reach2(X') :-
  reach1(X), R1(X,X').
reach2(X') :-
  reach2(X), R2(X,X').
reach2(X') :-
  reach3(X), R3(X,X').

reach3(X') :-
  reach2(X), R2(X,X').
reach3(X') :-
  reach3(X), R3(X,X').
```

Horn clause abstraction of product



reach11(X,Y) :- initA(X), initB(Y).

reach22(X',Y') :-
  reach11(X,Y), R_{A1}(X,X'), R_{B1}(Y,Y').
reach22(X',Y') :-
  reach22(X,Y), R_{A2}(X,X'), R_{B2}(Y,Y').
reach22(X',Y') :-
  reach23(X,Y), R_{A2}(X,X'), R_{B3}(Y,Y').

reach23(X',Y') :-
  reach22(X,Y), R_{A2}(X,X'), R_{B2}(Y,Y').
reach23(X',Y') :-
  reach23(X,Y), R_{A2}(X,X'), R_{B3}(Y,Y').

Implicit representation of product

- The product is represented implicitly
- The transition relation for each product state is not explicitly computed
  - this would be computationally expensive

- We thus have control over the trade-off of precision and complexity
  - drawing on methods from Horn clause analysis and verification

Refinement of automaton

- Analysis of the product automaton Horn clauses yields approximation of the values in which each product state is reachable
- This can give more precise analysis of *may-run-in-parallel*
- Let Φ_{AB} be an approximation of the state in which product state AB is reachable.
  - Apply forward slicing using Φ_{AB} to states A and B
  - eliminate infeasible path combinations that cannot run in parallel

How far have we got?

- ✓ Construction of thread automata
  - ✓ Prototype shown in Madrid November 2014.
    - ❖ current work – extension of the language and semantics
    - ✓ now have AST from the real XC parser (thanks Jamie)
    - ❖ construction of product automata
- ✓ Analysis and refinement of state relations $R_S(X,X')$
    - ✓ tools for Horn clause abstract interpretation
    - ✓ tools for slicing Horn clauses
- ❖ Complexity analysis of states
    - ❖ integration with CiaoPP resource analysis (and WCET?)
    - ❖ other recent techniques for cost analysis (CAV'2014)

ENTRA

Future

- Integration with timing analysis?
- Estimation of throughput, frequency of communication, idle time, etc.
- Handling master-slave and stream communication in XC.

- Energy analysis of whole program
  - ✓ single-threaded energy analysis of thread automaton states
  - ✓ uses estimates of number of active threads as needed by the Bristol energy model