Parallel Algorithms for Predicate Detection

Vijay Garg, Rohan Garg

Department of Electrical and Computer Engineering,
The University of Texas at Austin,
Austin, TX 78712

Motivation

Distributed and concurrent programs are prone to errors

Debugging and Testing:

- Traces need to be analyzed to locate bugs.

Software Quality Assurance:

- Can I trust the results of the computation? Does it satisfy all required properties?

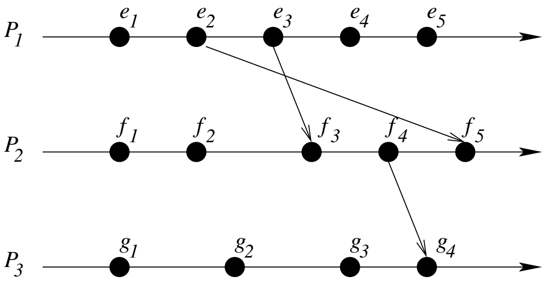
Outline of the Talk

- Predicate Detection Problem
- Parallel Algorithm for Detecting Conjunctive Predicates
- Parallel Algorithms for Detecting Data Race Predicate

Modeling a Distributed Computation

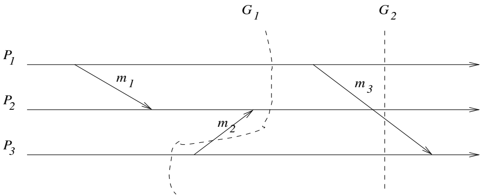
A computation is $(E, \rightarrow)$ where E is the set of events and $\rightarrow$ (happened-before) is the smallest relation that includes:

- e occurred before f in the same process implies $e \rightarrow f$.
- e is a send event and f the corresponding receive implies $e \rightarrow f$.
- if there exists g such that $e \rightarrow g$ and $g \rightarrow f$, then $e \rightarrow f$.



[Lamport 78]

Consistent Global State (CGS) of a Distributed System



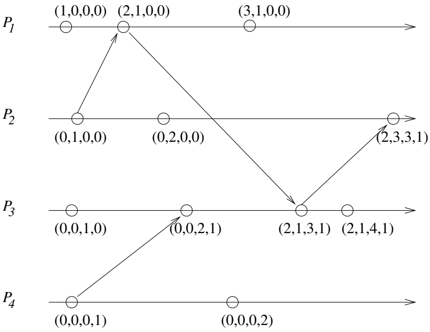
Consistent global state = subset of events executed so far

A subset G of E is a consistent global state (also called a consistent cut) if

$\forall e, f \in E : (f \in G) \wedge (e \rightarrow f)(e \in G)$

Tracking Dependency

Problem: Given $(E, \rightarrow)$, assign timestamps v to events in E such that $\forall e, f \in E : e \rightarrow f \equiv v(e) < v(f)$



Timestamps: Vector Clocks [Fidge 89, Mattern 89]:

Global Predicate Detection Problem

- Input:
  traces of n processes $P_1, \ldots, P_n$,
  (a trace is a sequence of vector clocks with relevant state information)
  B: boolean predicate,
- Output:
  (yes, G), if there exists a consistent global state G such that $B(G)$;
  no, if there does not exists any global state satisfying B

Detecting B is NP-complete even when B is a 2-CNF expression of local predicates [Chase and Garg 94, Mittal and Garg 01].

Outline of the Talk

- Predicate Detection Problem
- Parallel Algorithm for Detecting Conjunctive Predicates
- Parallel Algorithms for Detecting Data Race Predicate

Conjunctive Predicates

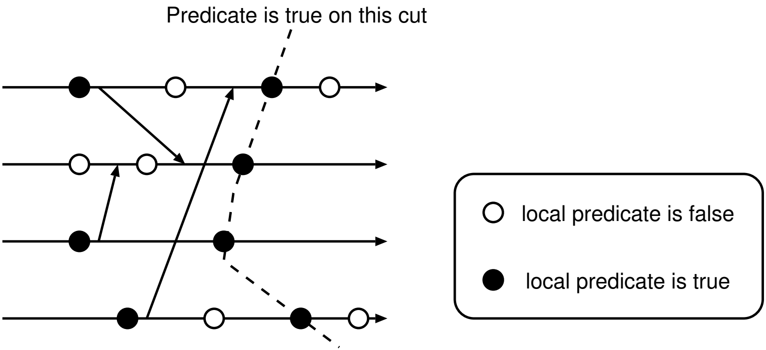
Local predicate:
predicate that can be evaluated by a process locally
Examples:
$(P_1 \text{ } is \text{ } in \text{ } CS)$,
(P_i is not in the leader mode)
Conjunctive Predicate:
A conjunction of local predicates
$B = l_1 \wedge l_2 \wedge \ldots \wedge l_n$
where each l_i is a local predicate
Examples:
$B \equiv (P_1 \text{ } is \text{ } in \text{ } CS) \wedge (P_2 \text{ } is \text{ } in \text{ } CS)$
$B \equiv (P_1 \text{ } is \text{ } not \text{ } leader) \wedge \ldots \wedge (P_n \text{ } is \text{ } not \text{ } leader)$

Importance of Conjunctive Predicates

Sufficient for detection of the following global predicates

- Any boolean expression in disjunctive normal form
  $B = B_1 \vee B_2 \vee \ldots B_k$
  where each B_i is a conjunction of local predicates
  Each conjunction B_i can be detected in parallel.
  Example: x, y and z are in three different processes. Then,
  $even(x) \wedge ((y < 0) \vee (z > 6))$
  $\equiv$
  $(even(x) \wedge (y < 0)) \vee (even(x) \wedge (z > 6))$
- predicate satisfied by only a small number of values
  Example: x and y are in different processes.
  $(x = y)$ is not a *local* predicate but x and y are binary.

Conditions for Conjunctive Predicates



Possibly $(l_1 \wedge l_2 \wedge \ldots l_n)$ is true **iff** there exist s_i in P_i such that l_i is true in state s_i, and s_i and s_j are incomparable for distinct i, j.

Related Work: Centralized Algorithm

[Garg and Waldecker 94] Send vector clocks to a checker process for events which satisfy local predicate in any message interval.

Checker process

1. Begin with the initial global state
2. Repeatedly eliminate any vector that happened before any other vector along the current global state.

Predicate is true for the first time

- all vectors in the current global state are pairwise concurrent

Predicate is false

- if we eliminate the final vector from any process

Work Complexity: $O(mn^2)$

n: number of processes,

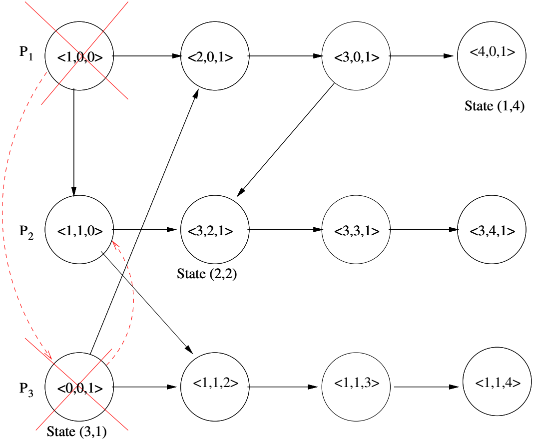
m: number of events per process

Main Result for Conjunctive Predicate Detection

Parallel Algorithm for Conjunctive Predicate Detection

- Theorem 1: The conjunctive predicate detection problem on n processes with at most m states per process can be solved in $O(\log mn)$ time using $O(m^3 n^3 \log mn)$ operations on the common CRCW PRAM.

Key Idea: State Rejection



- Rejection of state $< 1, 0, 0 > \Rightarrow$ advance to $< 2, 0, 1 >$
- Then, reject $< 0, 0, 1 >$ because $< 0, 0, 1 > \rightarrow < 2, 0, 1 >$

State Rejection Graph

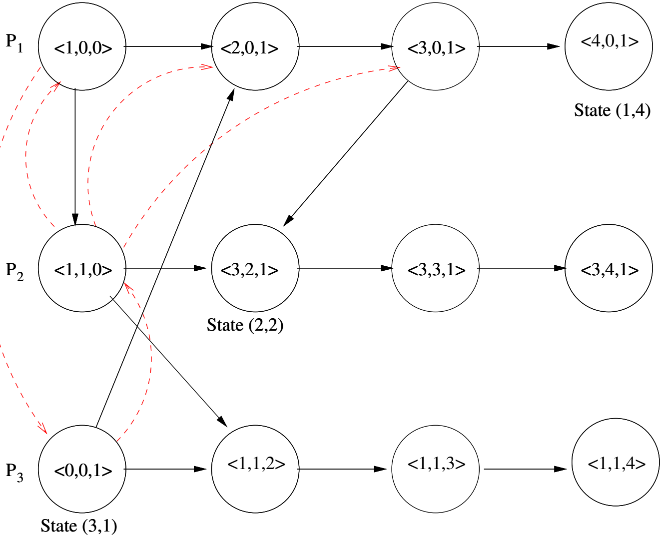


Figure: State Rejection Graph of a computation shown in dashed arrows

Parallel Algorithm

Step 1: Create F array

set of states rejected in the first round

Step 2:Create R matrix

State Rejection Graph // Adjacency Matrix

$R : [(1 \ldots n, 1 \ldots m), (1 \ldots n, 1 \ldots m)]$ of $0 \ldots 1$;

Step 3: Create RT matrix

$RT : array[(1 \ldots n, 1 \ldots m), (1 \ldots n, 1 \ldots m)]$ of $0 \ldots 1$;

$RT := TransitiveClosure(R)$;

Step 4: Create *valid* array

states reachable from F using RT

replace invalid states by 0

Step 5: Create *cut* array

first valid state on each process

Other Efficiently Detectable Predicates

- $x_1 \geq x_2$
  x_i on different processes, non-decreasing
- Communication Deadlock
  P_i is waiting for a message from P_j and P_j is waiting for a message from P_i and there is no in-transit message between them.
- All processes in phase 2
  All processes have started phase two and there is no in-transit message from phase one.
- Targeted virtual time
  All processes have local virtual time greater than 100 and there is no in-transit message with virtual time less than 100.

Outline of the Talk

- Predicate Detection Problem
- Parallel Algorithm for Detecting Conjunctive Predicates
- Parallel Algorithms for Detecting Data Race Predicate

Data Race Predicate Detection

Data Race Predicate Detection Problem: Given a multithreaded computation $(E, \rightarrow)$, is there any instance of a *read-write conflict*, or a *write-write conflict*.

Are there two concurrent events e and f in E such that they are on the same object and one of them is a write operation?

Data Race Predicate Detection

Brute Force Parallel Algorithm 1

for all events e and f in parallel do
  if $(e||f) \wedge ((e.op = write) \vee (f.op = write)) \wedge (e.object = f.object))$
    return "data race"
endfor;
return "no data race"

Time: $O(1)$, Work: $O(m^2 n^2)$
n: number of processes,
m: number of events per process

Data Race Detection Algorithm 2

Step 1: Compute projections for all objects

for all $(i \in [n], obj \in [q])$ in parallel do

$objectTrace[i][obj] := projection(trace[i], obj)$;

Step 2: Do binary search for each event and process

Time: $O(\log m)$, Work: $O(mn(n+q)\log m)$

n: number of processes,

m: number of events per process,

q: number of objects

Taking Projection

input inTrace: trace of a single process on multiple objects
obj: specific object
output outTrace: projection of the trace on the object *obj*

$loc : array[1..m]$ of integer;
for all $(k \in [m])$ in parallel do
if $(inTrace[k].object = obj)$ **then** $loc[k] := 1$
else $loc[k] := 0$;

$loc := parallelPrefixSum(loc)$;

for all $(k \in [m])$ in parallel do
if $(inTrace[k].object = obj)$ **then**
$outTrace[loc[k]] := inTrace[k]$

Time:O(log m); *Work* : $O(m)$

Data Race Detection Algorithm 2

Step 1: Compute projections for all objects

$objectTrace[j][obj]$ available for each j, obj

Step 2: Do binary search for each event and process

for all $(i \in [n], j \in [n], k \in [m])$ in parallel do

  if $(v[i][k].op = write)$

    $obj := v[i][k].object$;

    binary search $v[i][k]$ in $objectTrace[j][obj]$

    if (found incomparable vector)

    return "data race exists" for $v[i][k].object$;

endfor;

return "no data race"

Time: $O(\log m)$, Work: $O(mn(n+q)\log m)$

Data Race Detection Algorithm 3

Reducing work to $O(mnq \log mn \log n)$

Step 1: Merge traces for writes for every object

$L :=$ set of n traces each with m vectors;

for $obj \in 1 \ldots q$ in parallel do

$L_{obj} := L$ projected on obj and write operations;

$mergedTrace_{obj} :=$ Algo Mutex-Detect applied to L_{obj};

if (incomparable vectors found) return "write-write data race";

Step 2: Do binary search for all read operations

if (incomparable vectors found)

return "read-write data race";

return "no data race"

Time: $O(\log mn \log n)$, Work: $O(mnq \log mn \log n)$

Algorithm Mutex-Detect

```
L := set of n traces each with m vectors;
numTraces := n; // assume n is a power of 2
for  r := 1 ... log n do // in sequence
    // parallel merge trace 2j with trace 2j + 1
    for all u in trace 2j and 2j + 1 in parallel do
        rank1:= binary search u in trace[2j]; // rank of u in trace[2j]
        rank2 := binary search u in trace[2j + 1]; // rank in trace[2j +
        if (binary search finds incomparable vector)
            return  "incomparable vectors"
        else write u at rank1 + rank2 in the merged trace;
    numTraces := numTraces/2;
```

Time: $O(\log mn \log n)$, Work: $O(mn \log mn \log n)$

Data Race Detection Algorithm 3: Step 1

Step 1: Merge traces for writes for every object

$L :=$ set of n traces each with m vectors;

for $obj \in 1 \ldots q$ in parallel do

$L_{obj} := L$ projected on obj and write operations;

$mergedTrace_{obj} :=$ Algo Mutex-Detect applied to L_{obj};

if (incomparable vectors found) return "write-write data race";

Step 2: Do binary search for all read operations

if (incomparable vectors found)

return "read-write data race";

return "no data race"

Time: $O(\log mn \log n)$, Work: $O(mnq \log mn \log n)$

Data Race Detection Algorithm 3: Step 2

Step 1: Merge traces for writes for every object

if (incomparable vectors found) return "write-write data race";

Step 2: Do binary search for all read operations

for all $(i \in [n], k \in [m])$ in parallel do

if $(v[i][k].op = read) \wedge (v[i][k].object = obj)$;

binary search $v[i][k]$ in $mergedTrace_{obj}$

if (incomparable vectors found)

return "read-write data race";

return "no data race"

Time: $O(\log mn \log n)$, Work: $O(mnq \log mn \log n)$

Main Result for Data Race Predicate Detection

Theorem 2:

1. There exists a parallel algorithm that detects the data race predicate in $O(1)$ time and $O(m^2n^2)$ work using $O(m^2n^2)$ processors on the CREW PRAM.
2. There exists a parallel algorithm that detects the data race predicate in $O(\log m)$ time and $O(mn(n+q)\log m)$ work using $O(mn^2)$ processors on the CREW PRAM.
3. There exists a parallel algorithm that detects the data race predicate in $O(\log mn \log n)$ time and $O(mnq \log mn \log n)$ work using $O(mnq)$ processors on the CREW PRAM.

n: number of processes,

m: number of events per process,

q: number of objects

Future Work

- Reducing work complexity of Conjunctive Predicate Detection
  Parallel Time complexity: $O(\log mn)$
  Parallel Work Complexity: $O(m^3 n^3 \log mn)$
  Sequential Algorithm Complexity: $O(mn^2)$
- Reducing work complexity of Date Race Predicate Detection:
  Parallel Time complexity: $O(\log mn \log n)$
  Parallel Work Complexity: $O(mnq \log mn \log n)$
  Sequential Algorithm Complexity: $O(mn \log mn)$
- Parallel Algorithm for Linear Predicates
  Finding the man-optimal stable matching is equivalent to detecting linear predicates (a generalization of conjunctive predicates). Are detecting linear predicates in NC?