Deterministic Galois: On-demand, Portable and Parameterless

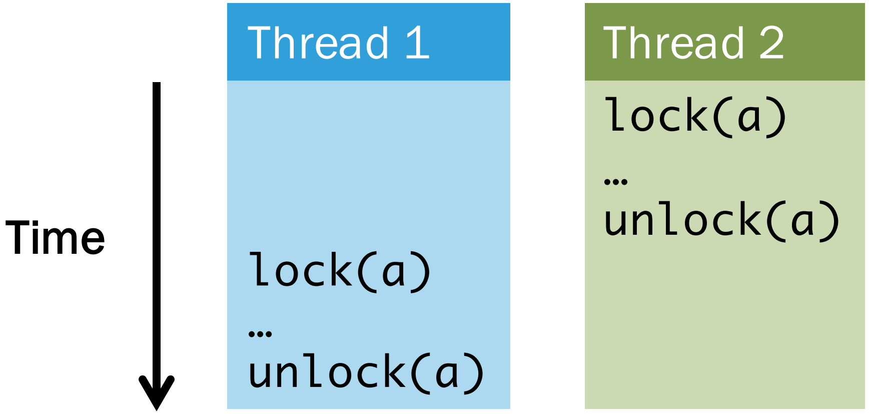
Donald Nguyen

Andrew Lenharth, Keshav Pingali

The University of Texas at Austin

Why Determinism?

- Parallelism introduces non-determinism from scheduling



```
lock(a)
…
unlock(a)
```

```
lock(a)
…
unlock(a)
```

- Goal: Eliminate scheduling non-determinism
  - Simplify debugging, fault tolerance

Desired Qualities

- On-demand
  - Determinism can be expensive; allow users to easily enable determinism as desired
- Portable
  - Deterministic result should be the same regardless of machine architecture, including number of threads
- Parameterless
  - There should be no user-tunable parameters that affect output

Avenues to Determinism

Determinism by Construction

- Programs must conform to set of deterministic program constructs
- Examples
  - Fork-join [Blumofe95]
  - DPJ [Bocchino11]
  - PBBS [Blelloch11]

Avenues to Determinism

Determinism by Construction

- Programs must conform to set of deterministic program constructs
- Examples
  - Fork-join [Blumofe95]
  - DPJ [Bocchino11]
  - PBBS [Blelloch11]

- On-demand?
- Portable?
- Parameterless?

Avenues to Determinism

Determinism by Construction

- Programs must conform to set of deterministic program constructs
- Examples
  - Fork-join [Blumofe95]
  - DPJ [Bocchino11]
  - PBBS [Blelloch11]

- On-demand? ☒
- Portable?
- Parameterless?

Avenues to Determinism

Determinism by Construction

- Programs must conform to set of deterministic program constructs
- Examples
  - Fork-join [Blumofe95]
  - DPJ [Bocchino11]
  - PBBS [Blelloch11]

- On-demand? ☒
- Portable? ☑
- Parameterless?

Avenues to Determinism

Determinism by Construction

- Programs must conform to set of deterministic program constructs
- Examples
  - Fork-join [Blumofe95]
  - DPJ [Bocchino11]
  - PBBS [Blelloch11]

- On-demand? ☒
- Portable? ☑
- Parameterless? ?

Avenues to Determinism

Determinism by Construction

- Programs must conform to set of deterministic program constructs
- Examples
  - Fork-join [Blumofe95]
  - DPJ [Bocchino11]
  - PBBS [Blelloch11]

Determinism by Scheduling

- Provide deterministic version of low-level scheduling primitives
  - Determinism w/o rewriting programs
- Examples
  - Kendo [Olszewski09]
  - RCDC [Devietti11]

- On-demand? ☒
- Portable? ☑
- Parameterless? ?

Avenues to Determinism

Determinism by Construction

- Programs must conform to set of deterministic program constructs
- Examples
  - Fork-join [Blumofe95]
  - DPJ [Bocchino11]
  - PBBS [Blelloch11]

- On-demand? ☒
- Portable? ☑
- Parameterless? ?

Determinism by Scheduling

- Provide deterministic version of low-level scheduling primitives
  - Determinism w/o rewriting programs
- Examples
  - Kendo [Olszewski09]
  - RCDC [Devietti11]

- On-demand?
- Portable?
- Parameterless?

Avenues to Determinism

Determinism by Construction

- Programs must conform to set of deterministic program constructs
- Examples
  - Fork-join [Blumofe95]
  - DPJ [Bocchino11]
  - PBBS [Blelloch11]

- On-demand? ☒
- Portable? ☑
- Parameterless? ?

Determinism by Scheduling

- Provide deterministic version of low-level scheduling primitives
  - Determinism w/o rewriting programs
- Examples
  - Kendo [Olszewski09]
  - RCDC [Devietti11]

- On-demand? ☑
- Portable?
- Parameterless?

Avenues to Determinism

Determinism by Construction

- Programs must conform to set of deterministic program constructs
- Examples
  - Fork-join [Blumofe95]
  - DPJ [Bocchino11]
  - PBBS [Blelloch11]

- On-demand? ☒
- Portable? ☑
- Parameterless? ?

Determinism by Scheduling

- Provide deterministic version of low-level scheduling primitives
  - Determinism w/o rewriting programs
- Examples
  - Kendo [Olszewski09]
  - RCDC [Devietti11]

- On-demand? ☑
- Portable? ☒
- Parameterless?

Avenues to Determinism

Determinism by Construction

- Programs must conform to set of deterministic program constructs
- Examples
  - Fork-join [Blumofe95]
  - DPJ [Bocchino11]
  - PBBS [Blelloch11]

- On-demand? ☒
- Portable? ☑
- Parameterless? ?

Determinism by Scheduling

- Provide deterministic version of low-level scheduling primitives
  - Determinism w/o rewriting programs
- Examples
  - Kendo [Olszewski09]
  - RCDC [Devietti11]

- On-demand? ☑
- Portable? ☒
- Parameterless? ?

Avenues to Determinism

High-level Non-deterministic Programming Model

- Deterministic Galois (this work)
- Non-determinism expressed in constructs beyond threads and locks
  - System responsible for deterministic execution if desired
- Targeted towards fine-grain tasks
  - 10--1000 cycles per task
  - Frequent communication

Avenues to Determinism

High-level Non-deterministic Programming Model

- Deterministic Galois (this work)
- Non-determinism expressed in constructs beyond threads and locks
  - System responsible for deterministic execution if desired
- Targeted towards fine-grain tasks
  - 10--1000 cycles per task
  - Frequent communication

- On-demand?
- Portable?
- Parameterless?

Avenues to Determinism

High-level Non-deterministic Programming Model

- Deterministic Galois (this work)
- Non-determinism expressed in constructs beyond threads and locks
  - System responsible for deterministic execution if desired
- Targeted towards fine-grain tasks
  - 10--1000 cycles per task
  - Frequent communication

- On-demand? ☑
- Portable?
- Parameterless?

Avenues to Determinism

High-level Non-deterministic Programming Model

- Deterministic Galois (this work)
- Non-determinism expressed in constructs beyond threads and locks
  - System responsible for deterministic execution if desired
- Targeted towards fine-grain tasks
  - 10--1000 cycles per task
  - Frequent communication

- On-demand? ☑
- Portable? ☑
- Parameterless?

Avenues to Determinism

High-level Non-deterministic Programming Model

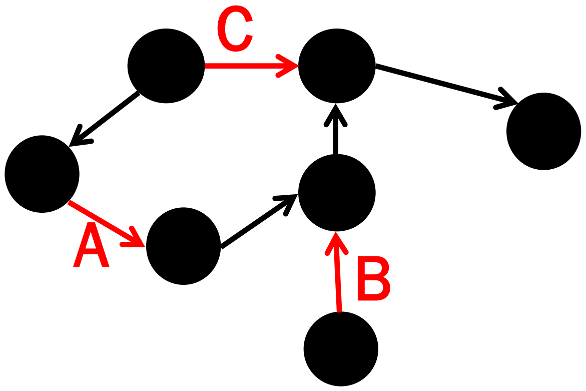
- Deterministic Galois (this work)
- Non-determinism expressed in constructs beyond threads and locks
  - System responsible for deterministic execution if desired
- Targeted towards fine-grain tasks
  - 10--1000 cycles per task
  - Frequent communication

- On-demand? ☑
- Portable? ☑
- Parameterless? ☑

Galois Programming Model

- Galois system: runtime and library of concurrent data structures
- Set iterators express parallelism
  - Operator: function to apply
  - Neighborhood: data accessed
  - Cautious: operators read their entire neighborhood before writing to any element
    - Failsafe point: Point between reading and writing
- Set iterator produces some serialization of operator invocations
  - Only source of non-determinism in Galois programs

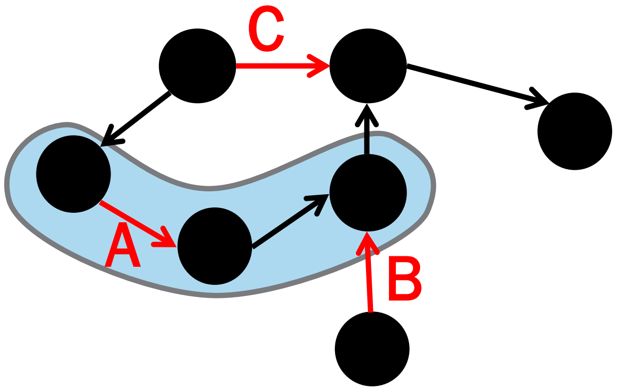
```
Graph g
for_each (Edge e : wl)
  Node n = g.getEdgeDst(e)
  …
  for (Node m :
g.out_edges(n))
    … g.getData(m) …
  …
```



Galois Programming Model

- Galois system: runtime and library of concurrent data structures
- Set iterators express parallelism
  - Operator: function to apply
  - Neighborhood: data accessed
  - Cautious: operators read their entire neighborhood before writing to any element
    - Failsafe point: Point between reading and writing
- Set iterator produces some serialization of operator invocations
  - Only source of non-determinism in Galois programs

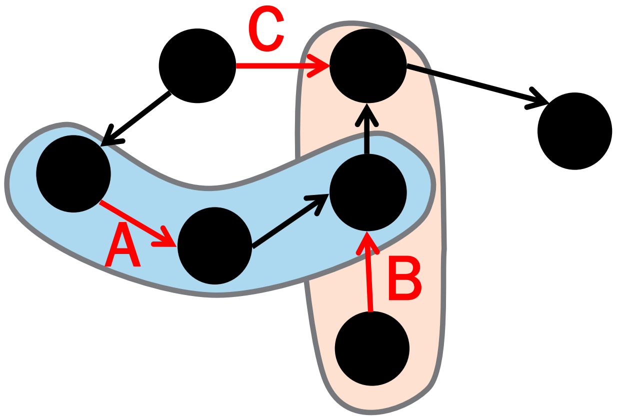
```
Graph g
for_each (Edge e : wl)
 Node n = g.getEdgeDst(e)
 …
 for (Node m :
g.out_edges(n))
  … g.getData(m) …
 …
```



Galois Programming Model

- Galois system: runtime and library of concurrent data structures
- Set iterators express parallelism
  - Operator: function to apply
  - Neighborhood: data accessed
  - Cautious: operators read their entire neighborhood before writing to any element
    - Failsafe point: Point between reading and writing
- Set iterator produces some serialization of operator invocations
  - Only source of non-determinism in Galois programs

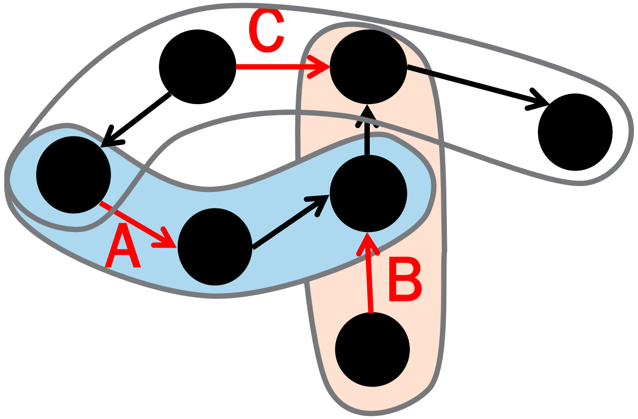
```
Graph g
for_each (Edge e : wl)
  Node n = g.getEdgeDst(e)
  …
  for (Node m :
g.out_edges(n))
    … g.getData(m) …
  …
```



Galois Programming Model

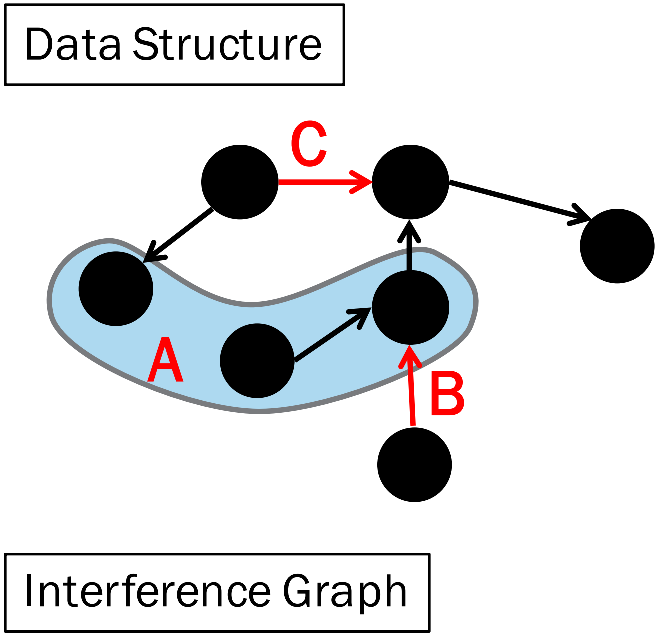
- Galois system: runtime and library of concurrent data structures
- Set iterators express parallelism
  - Operator: function to apply
  - Neighborhood: data accessed
  - Cautious: operators read their entire neighborhood before writing to any element
    - Failsafe point: Point between reading and writing
- Set iterator produces some serialization of operator invocations
  - Only source of non-determinism in Galois programs

```
Graph g
for_each (Edge e : wl)
  Node n = g.getEdgeDst(e)
  …
  for (Node m :
g.out_edges(n))
    … g.getData(m) …
  …
```



Deterministic Interference Graph Scheduling

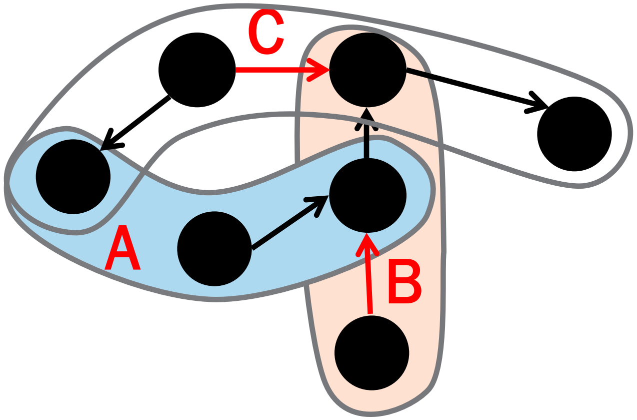
- Traditional Galois execution
  - Asynchronous, non-deterministic
- Deterministic Galois execution
  - Construct an interference graph
  - Execute independent set
  - Repeat



Deterministic Interference Graph Scheduling

- Traditional Galois execution
  - Asynchronous, non-deterministic
- Deterministic Galois execution
  - Construct an interference graph
  - Execute independent set
  - Repeat

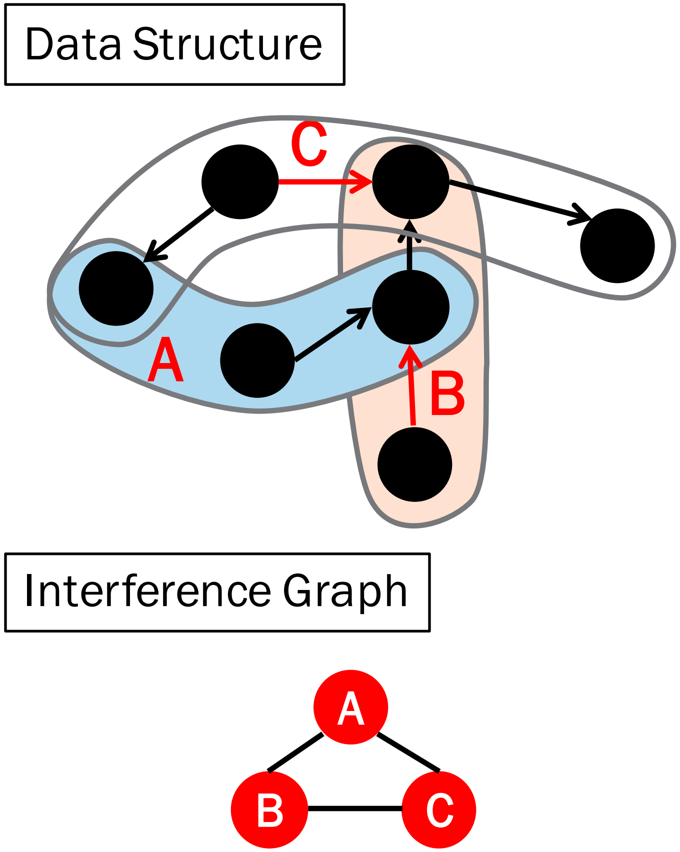
Data Structure



Interference Graph

Deterministic Interference Graph Scheduling

- Traditional Galois execution
  - Asynchronous, non-deterministic
- Deterministic Galois execution
  - Construct an interference graph
  - Execute independent set
  - Repeat



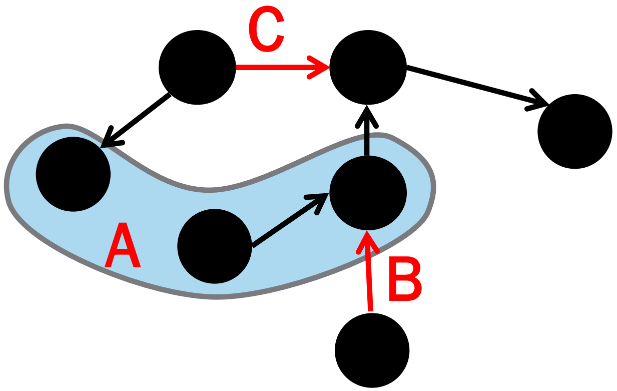
Data Structure

Interference Graph

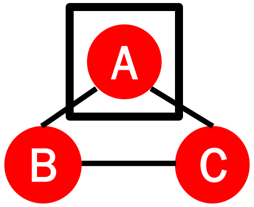
Deterministic Interference Graph Scheduling

- Traditional Galois execution
  - Asynchronous, non-deterministic
- Deterministic Galois execution
  - Construct an interference graph
  - Execute independent set
  - Repeat

Data Structure

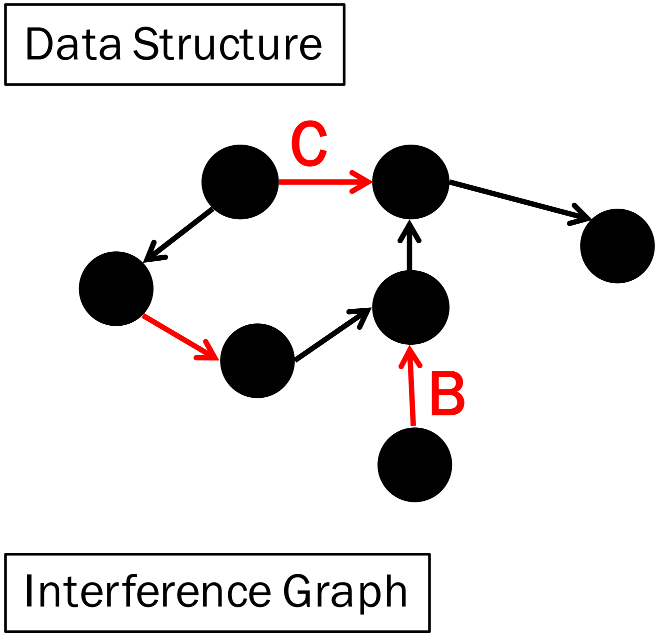


Interference Graph



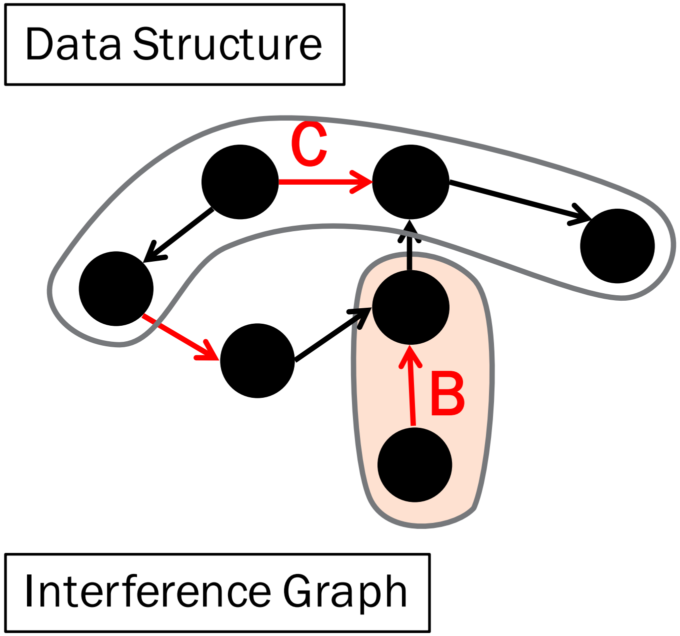
Deterministic Interference Graph Scheduling

- Traditional Galois execution
  - Asynchronous, non-deterministic
- Deterministic Galois execution
  - Construct an interference graph
  - Execute independent set
  - Repeat



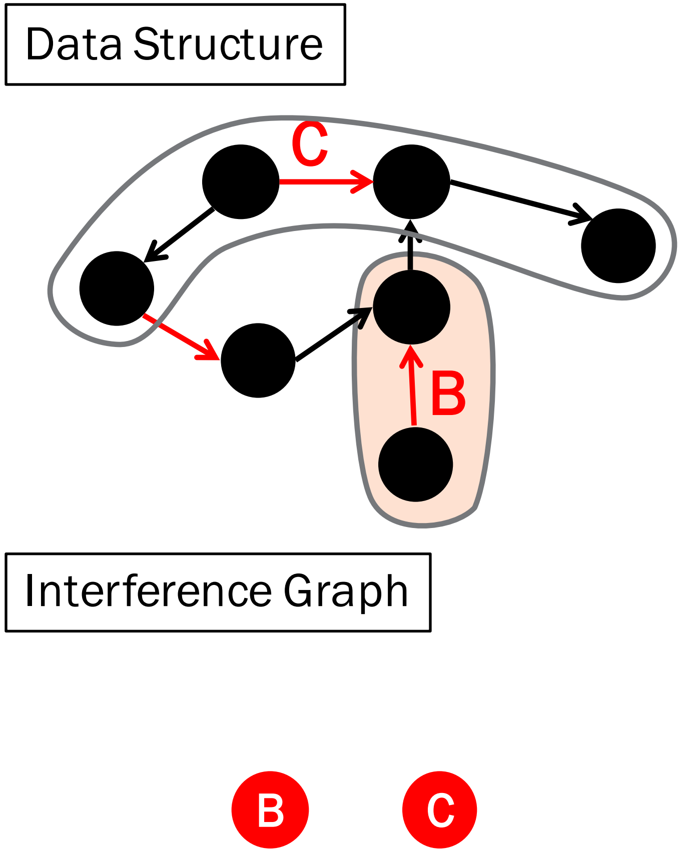
Deterministic Interference Graph Scheduling

- Traditional Galois execution
  - Asynchronous, non-deterministic
- Deterministic Galois execution
  - Construct an interference graph
  - Execute independent set
  - Repeat



Deterministic Interference Graph Scheduling

- Traditional Galois execution
  - Asynchronous, non-deterministic
- Deterministic Galois execution
  - Construct an interference graph
  - Execute independent set
  - Repeat

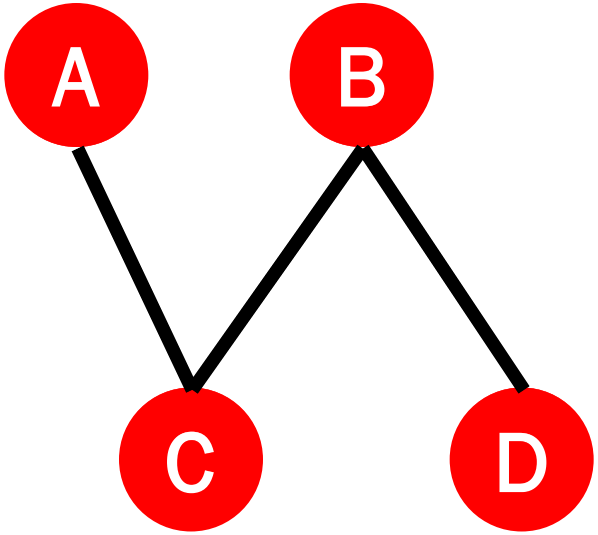


Data Structure

Interference Graph

DIG Scheduling

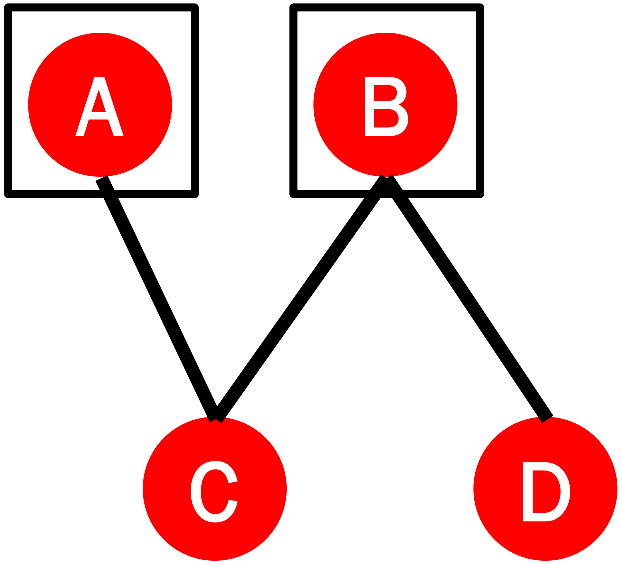
- Deterministically select independent set
  - Form total order on tasks
  - Select tasks that are least among direct neighbors



A < B < C < D

DIG Scheduling

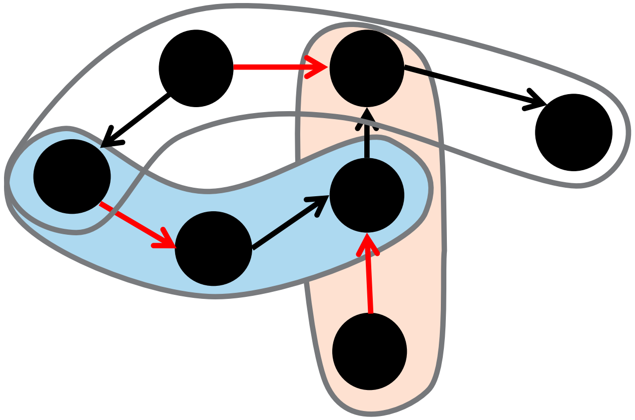
- Deterministically select independent set
  - Form total order on tasks
  - Select tasks that are least among direct neighbors



A < B < C < D

DIG Scheduling in Practice

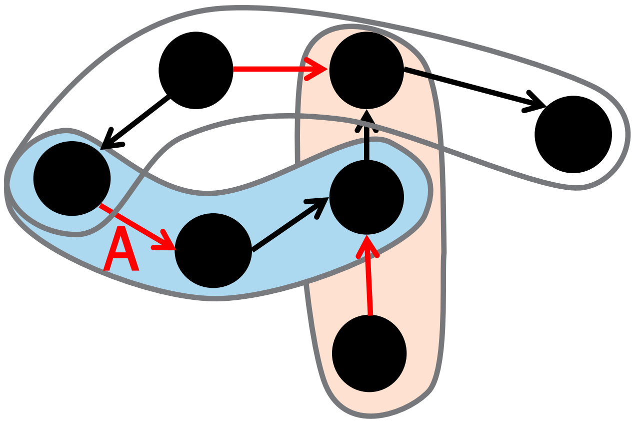
- Implicitly build neighborhoods and interference graph
- Marks associated with data structure elements
- Acquire marks by writing task id atomically
  - Overwriting an id only if replacing greater value
- Execute tasks whose neighborhood only contains their own marks
- Final mark values are deterministic



A < B < C

DIG Scheduling in Practice

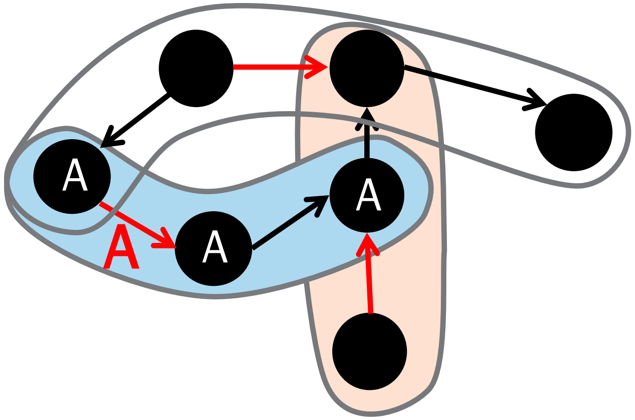
- Implicitly build neighborhoods and interference graph
- Marks associated with data structure elements
- Acquire marks by writing task id atomically
  - Overwriting an id only if replacing greater value
- Execute tasks whose neighborhood only contains their own marks
- Final mark values are deterministic



A < B < C

DIG Scheduling in Practice

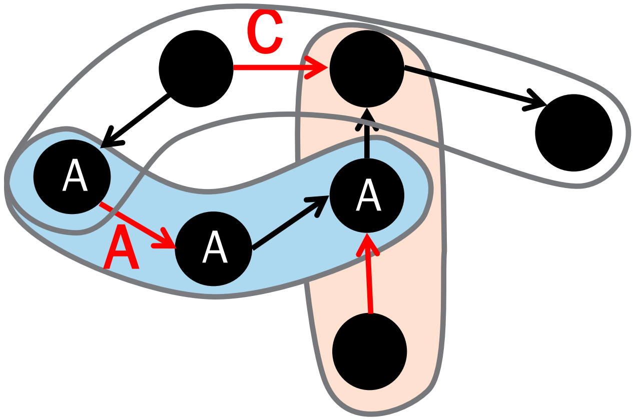
- Implicitly build neighborhoods and interference graph
- Marks associated with data structure elements
- Acquire marks by writing task id atomically
  - Overwriting an id only if replacing greater value
- Execute tasks whose neighborhood only contains their own marks
- Final mark values are deterministic



A < B < C

DIG Scheduling in Practice

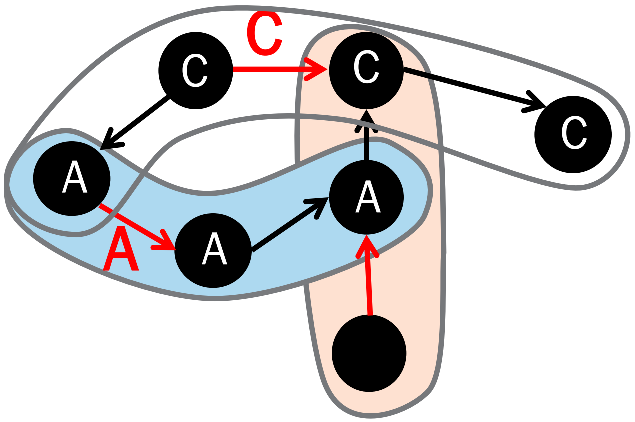
- Implicitly build neighborhoods and interference graph
- Marks associated with data structure elements
- Acquire marks by writing task id atomically
  - Overwriting an id only if replacing greater value
- Execute tasks whose neighborhood only contains their own marks
- Final mark values are deterministic



A < B < C

DIG Scheduling in Practice

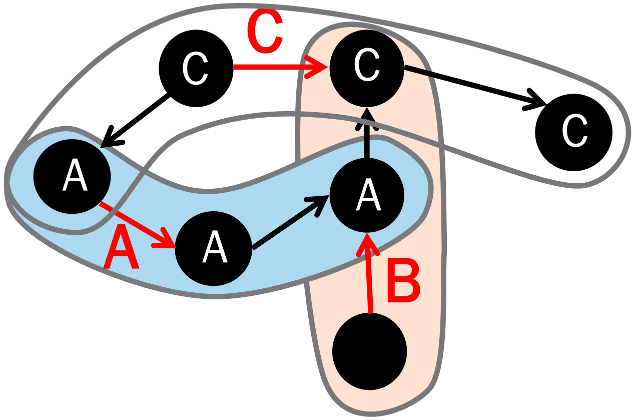
- Implicitly build neighborhoods and interference graph
- Marks associated with data structure elements
- Acquire marks by writing task id atomically
  - Overwriting an id only if replacing greater value
- Execute tasks whose neighborhood only contains their own marks
- Final mark values are deterministic



A < B < C

DIG Scheduling in Practice

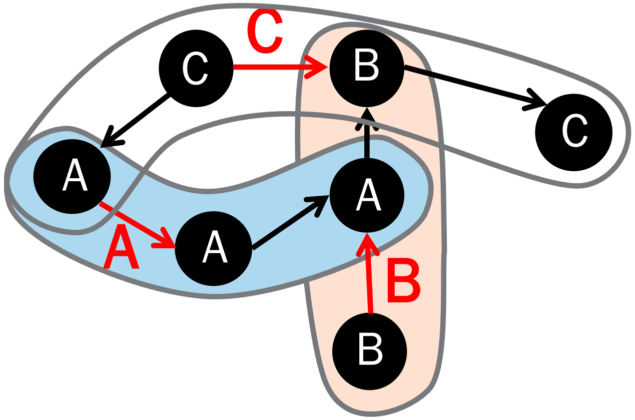
- Implicitly build neighborhoods and interference graph
- Marks associated with data structure elements
- Acquire marks by writing task id atomically
  - Overwriting an id only if replacing greater value
- Execute tasks whose neighborhood only contains their own marks
- Final mark values are deterministic



A < B < C

DIG Scheduling in Practice

- Implicitly build neighborhoods and interference graph
- Marks associated with data structure elements
- Acquire marks by writing task id atomically
  - Overwriting an id only if replacing greater value
- Execute tasks whose neighborhood only contains their own marks
- Final mark values are deterministic



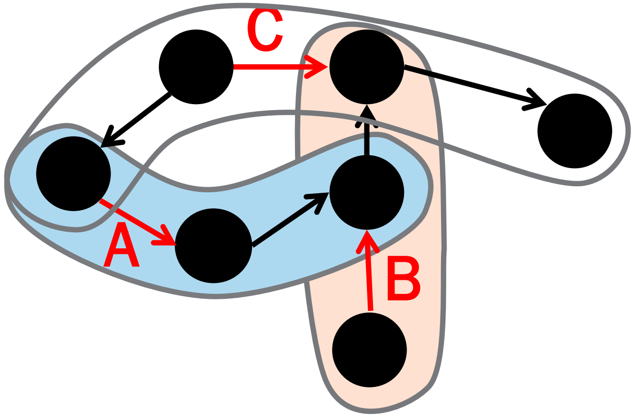
A < B < C

DIG Scheduling in Practice

- Sequence of rounds
  - Round has two phases
- Phase 1: Inspect neighborhoods
  - Execute operator to its failsafe point
  - Acquire marks
- Phase 2: Execute roots
  - Reexecute operator, checking mark values
  - Postpone any task that did not read its marks

```
for_each (Edge e : wl)
  Node n = g.getEdgeDst(e)
  …
  for (Node m :
g.out_edges(n))
    g.getData(m).value += 1
  …
```

Failsafe Point

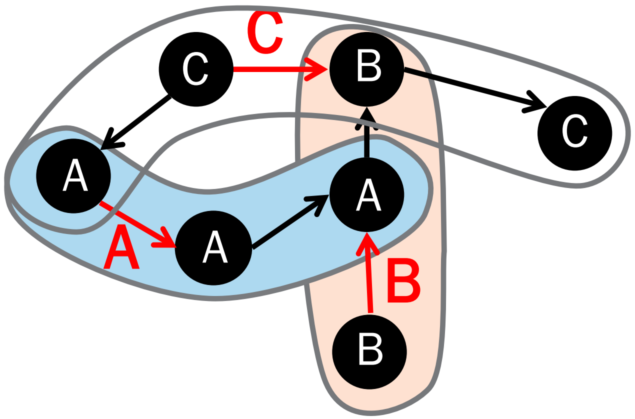


DIG Scheduling in Practice

- Sequence of rounds
  - Round has two phases
- Phase 1: Inspect neighborhoods
  - Execute operator to its failsafe point
  - Acquire marks
- Phase 2: Execute roots
  - Reexecute operator, checking mark values
  - Postpone any task that did not read its marks

```
for_each (Edge e : wl)
  Node n = g.getEdgeDst(e)
  …
  for (Node m :
g.out_edges(n))
    g.getData(m).value += 1
  …
```

Failsafe Point



Optimizations

- Resuming tasks
  - Avoid reexecuting tasks
  - Suspend and resume execution at failsafe point
  - Provide buffers to save local state
- Windowing
  - Inspect only a subset of tasks at a time
  - Adaptive algorithm varies window size

Evaluation

Platform

- Intel 4x10 core machine

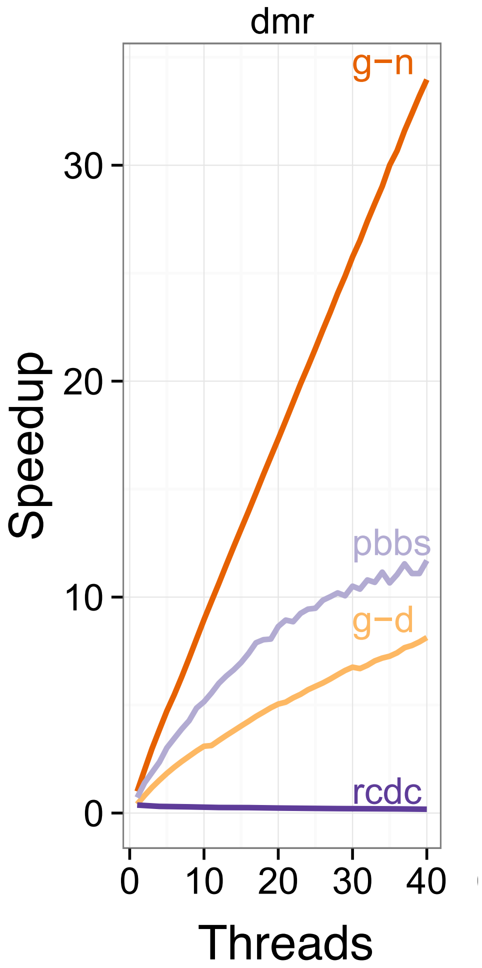
Applications

- PBBS [Blelloch11]
  - Breadth-first search (bfs)
  - Delaunay mesh refinement (dmr)
  - Delaunay triangulation (dt)
  - Maximal independent set (mis)

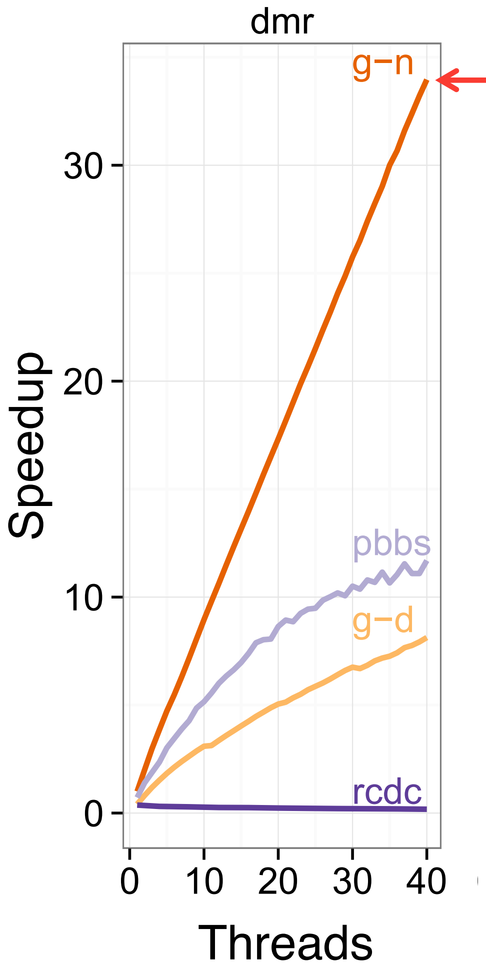
Deterministic Systems

- RCDC
  - Deterministic by scheduling
  - Implementation of Kendo algorithm
- PBBS
  - Deterministic by construction
  - Handwritten deterministic implementations
- Deterministic Galois
  - Non-deterministic programs (g-n) automatically made deterministic (g-d)

Evaluation

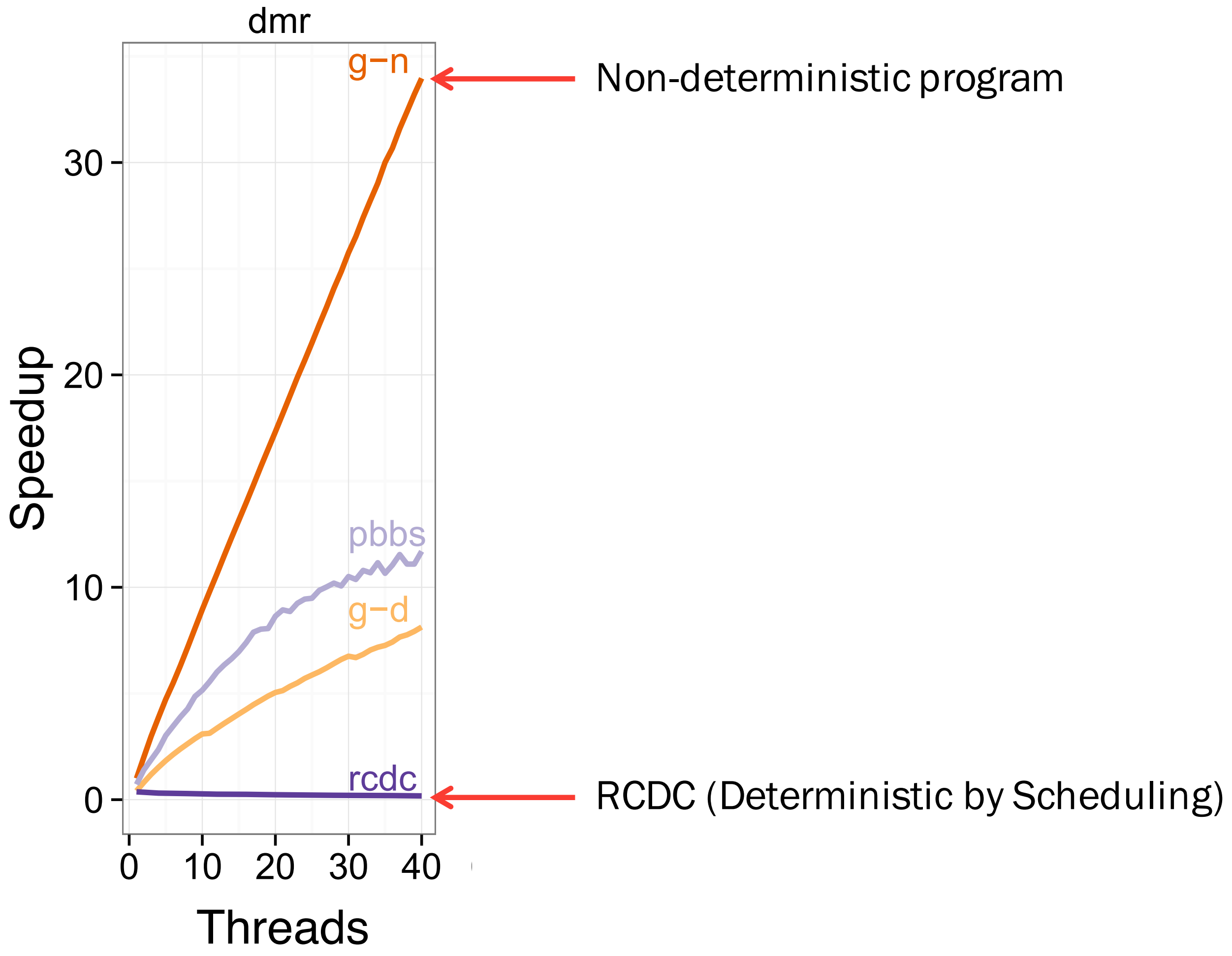


Evaluation



Non-deterministic program

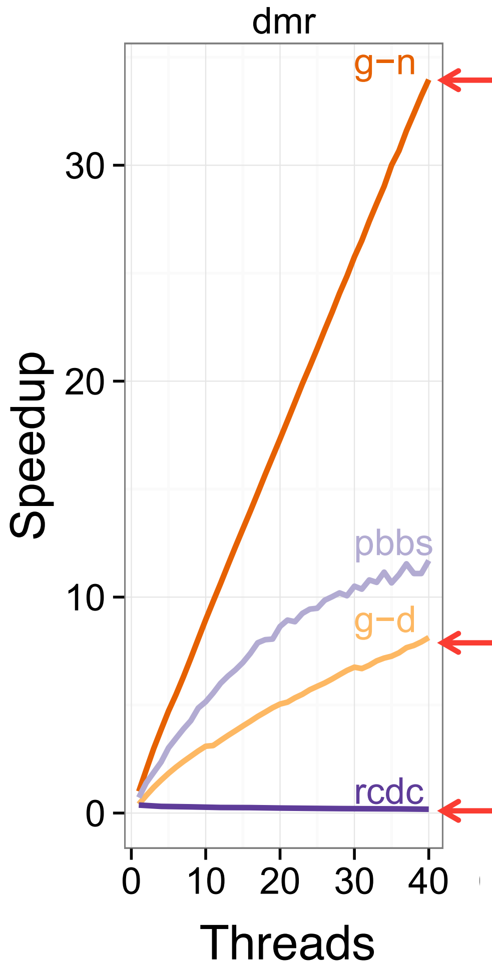
Evaluation



Non-deterministic program

RCDC (Deterministic by Scheduling)

Evaluation

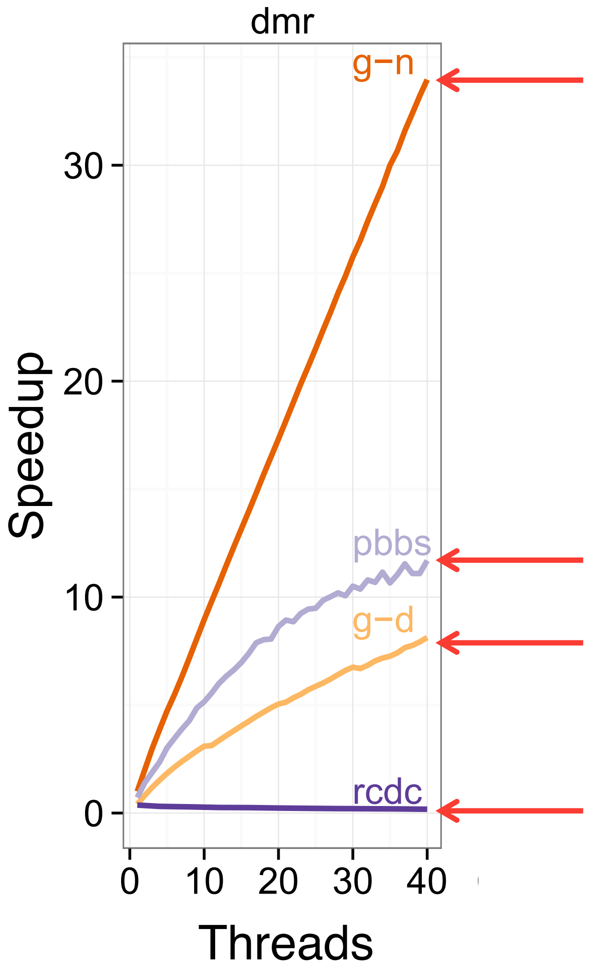


Non-deterministic program

Generated deterministic program

RCDC (Deterministic by Scheduling)

Evaluation



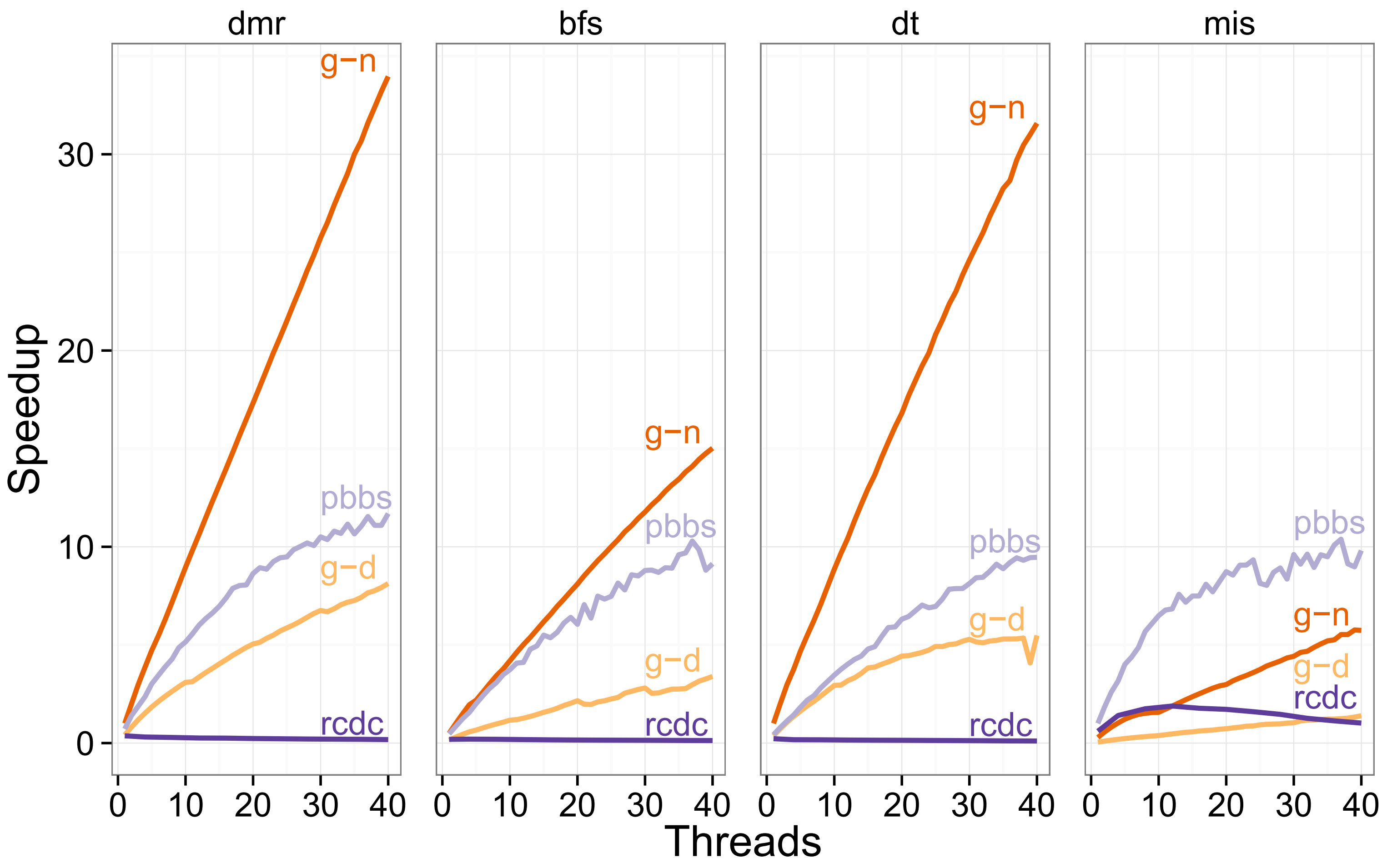
Non-deterministic program

PBBS (Deterministic by Construction)

Generated deterministic program

RCDC (Deterministic by Scheduling)

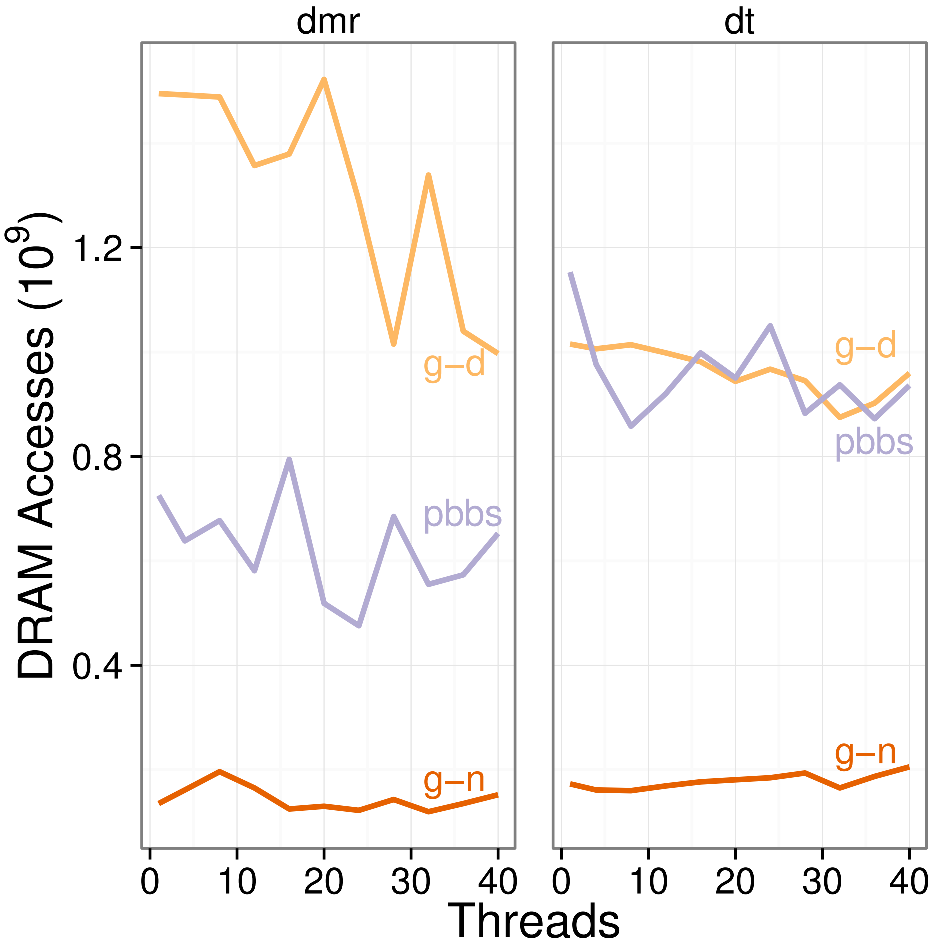
Evaluation


dmr
bfs
dt
mis
Speedup
0
10
20
30
Threads
0
10
20
30
40
g−n
pbbs
g−d
rcdc

Sources of Overhead

- Additional instructions
- Extending critical path by round execution
- Deterministic schedule chosen may not be optimal
- Locality
  - Inspecting a task and executing it are separated in time

Locality


dmr
dt
DRAM Accesses (10^9)
1.2
0.8
0.4
g–d
pbbs
g–n
g–d
pbbs
g–n
0
10
20
30
40
0
10
20
30
40
Threads

Conclusion

- Make programs deterministic using deterministic interference graph scheduling
  - Facilitates on-demand, portable and parameterless deterministic programs
  - Generated deterministic programs comparable to handwritten ones
  - Non-deterministic programs often much faster than deterministic ones
- In the paper
  - More applications, machines
  - Quantify locality under different systems
  - Measure impact of optimizations

http://iss.ices.utexas.edu/galois

Deterministic Galois: On-demand, Portable and Parameterless

Donald Nguyen

Andrew Lenharth, Keshav Pingali

The University of Texas at Austin