Executing Dynamic Data-Graph Computations Deterministically Using Chromatic Scheduling

Student presenter: Nikola Samardzic

Motivation

- Data graph computation is useful
- Dynamic data graph computation is even more useful
    - Dynamic data graph := active set depends on data
- Parallelism is good
- Determinism is good
    - Debugging (and even performance!)
- We want it all!

Key Contribution

- *Deterministic* execution of *dynamic* data graph computations

Serial Baseline

SERIAL-DDGC(G, f, Q_0)

for $v \in Q_0$
    ENQUEUE(Q, v)
$r = 0$
ENQUEUE(Q, NIL) // Sentinel NIL denotes the end of a round.
while $Q \neq \{\text{NIL}\}$
    $v =$ DEQUEUE(Q)
    if $v == \text{NIL}$
        $r \mathrel{+}= 1$
        ENQUEUE(Q, NIL)
    else
        $S = f(v)$
        for $u \in S$
            if $u \notin Q$
                ENQUEUE(Q, u)

f(v) returns the subset of v’s neighbors activated by the update

Serial Baseline

```
SERIAL-DDGC(G, f, Q_0)
for v ∈ Q_0
    ENQUEUE(Q, v)
r = 0
ENQUEUE(Q, NIL) // Sentinel NIL denotes the end of a round.
while Q ≠ {NIL}
    v = DEQUEUE(Q)
    if v == NIL
        r += 1
        ENQUEUE(Q, NIL)
    else
        S = f(v)
        for u ∈ S
            if u ∉ Q
                ENQUEUE(Q, u)
```

OK, so just make Q thread-safe, enqueue an entire round of active nodes, and run this for loop in parallel for each active node?

Determinism Aids Parallelism

- Nondeterministic programs are a pain to debug
- Locks suck
    - Implies lock acquisition and contention overheads
- An alternative: chromatic scheduling
    - Gives us determinism
    - And (basically) no locks!

Prism

```
PRISM(G, f, Q_0)
χ = COLOR-GRAPH(G)
r = 0
Q = Q_0
while Q ≠ ∅
    𝒞 = MB-COLLECT(Q)
    for C ∈ 𝒞
        parallel for v ∈ C
            active[v] = FALSE
            S = f(v)
            parallel for u ∈ S
                if CAS(active[u], FALSE, TRUE)
                    MB-INSERT(Q, u, color[u])
    r = r + 1
```

Prism

```
PRISM(G, f, Q_0)
  χ = COLOR-GRAPH(G)
  r = 0
  Q = Q_0
  while Q ≠ ∅
      𝒞 = MB-COLLECT(Q)
      for C ∈ 𝒞
          parallel for v ∈ C
              active[v] = FALSE
              S = f(v)
              parallel for u ∈ S
                  if CAS(active[u], FALSE, TRUE)
                      MB-INSERT(Q, u, color[u])
      r = r + 1
```

Coloring need not be perfect; ~solved problem

Prism

```
PRISM(G, f, Q₀)
χ = COLOR-GRAPH(G)
r = 0
Q = Q₀
while Q ≠ ∅
    𝒞 = MB-COLLECT(Q)
    for C ∈ 𝒞
        parallel for v ∈ C
            active[v] = FALSE
            S = f(v)
            parallel for u ∈ S
                if CAS(active[u], FALSE, TRUE)
                    MB-INSERT(Q, u, color[u])
    r = r + 1
```

Multibag funny business!

Prism

```
PRISM(G, f, Q_0)
  χ = COLOR-GRAPH(G)
  r = 0
  Q = Q_0
  while Q ≠ ∅
      𝒞 = MB-COLLECT(Q)
      for C ∈ 𝒞
          parallel for v ∈ C
              active[v] = FALSE
              S = f(v)
              parallel for u ∈ S
                  if CAS(active[u], FALSE, TRUE)
                      MB-INSERT(Q, u, color[u])
          r = r + 1
```

MB-Collect: Empty all bags from multibag Q into a collection C that is easy to iterate over

MB-Insert: Insert node u in bag color[u] of multibag Q

Prism

```
PRISM(G, f, Q_0)
χ = COLOR-GRAPH(G)
r = 0
Q = Q_0
while Q ≠ ∅
    C = MB-COLLECT(Q)
    for C ∈ C
        parallel for v ∈ C
            active[v] = FALSE
            S = f(v)
            parallel for u ∈ S
                if CAS(active[u], FALSE, TRUE)
                    MB-INSERT(Q, u, color[u])
        r = r + 1
```

Parallel compare and swap (CAS) only here to ensure each node receives at most one update per round

Prism

Questions?

```
PRISM(G, f, Q_0)
χ = COLOR-GRAPH(G)
r = 0
Q = Q_0
while Q ≠ ∅
    C = MB-COLLECT(Q)
    for C ∈ C
        parallel for v ∈ C
            active[v] = FALSE
            S = f(v)
            parallel for u ∈ S
                if CAS(active[u], FALSE, TRUE)
                    MB-INSERT(Q, u, color[u])
    r = r + 1
```

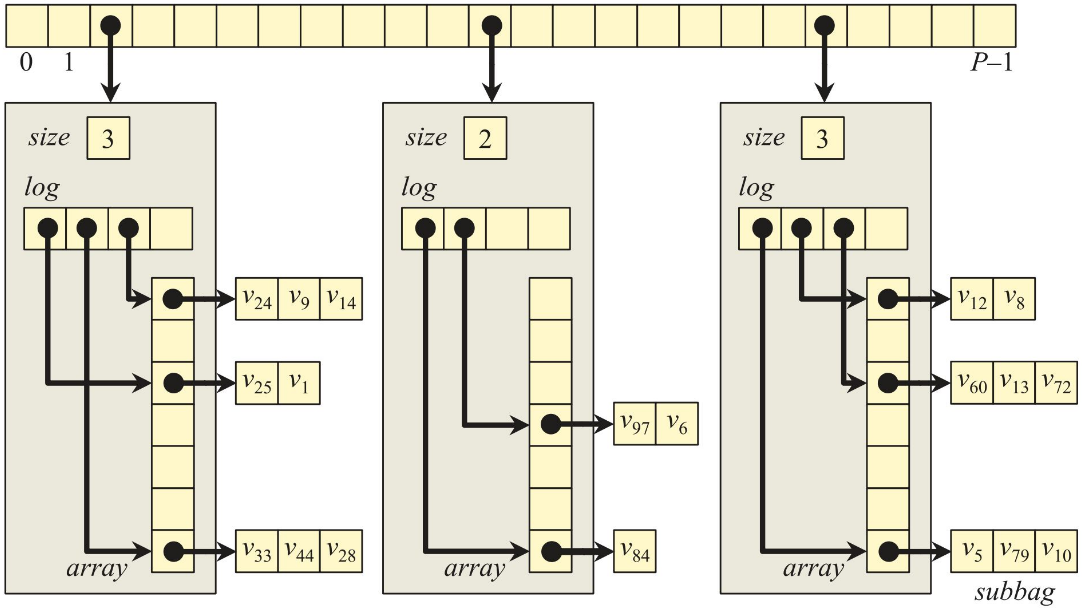
MB-Collect: Empty all bags from multibag Q into a collection C that is easy to iterate over

MB-Insert: Insert node u in bag color[u] of multibag Q

Key Idea: Parallel Lock-free Multibags

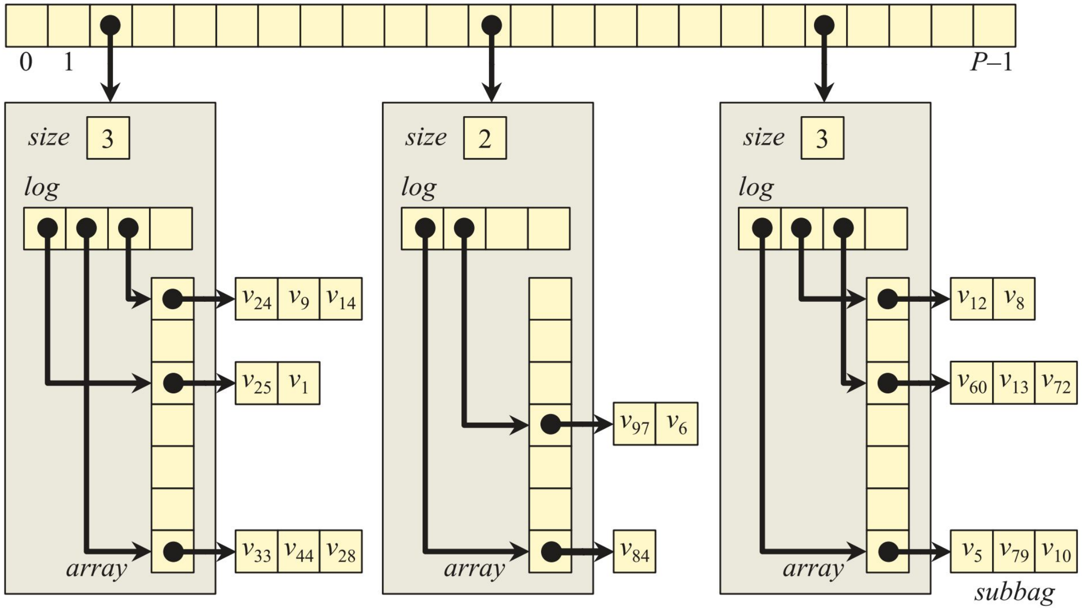
- Problem: How to make MB-Collect work-efficient?
- Just using over a bitmap of active nodes won’t do
    - Theta(V * Color) work to do a round; not work-efficient
- Just having each P maintain Color many local arrays of active elements also won’t do
    - Theta(P * Color) work to do a round; not work-efficient

Key Idea: Parallel Lock-free Multibags

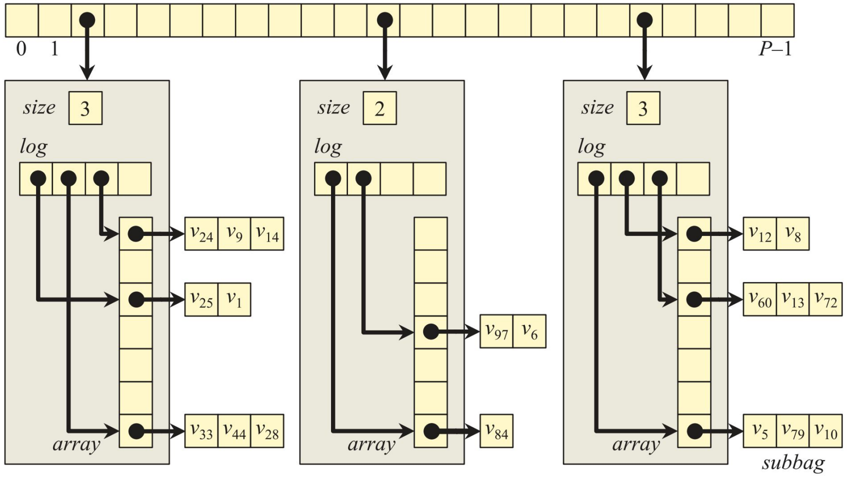


MB-Insert

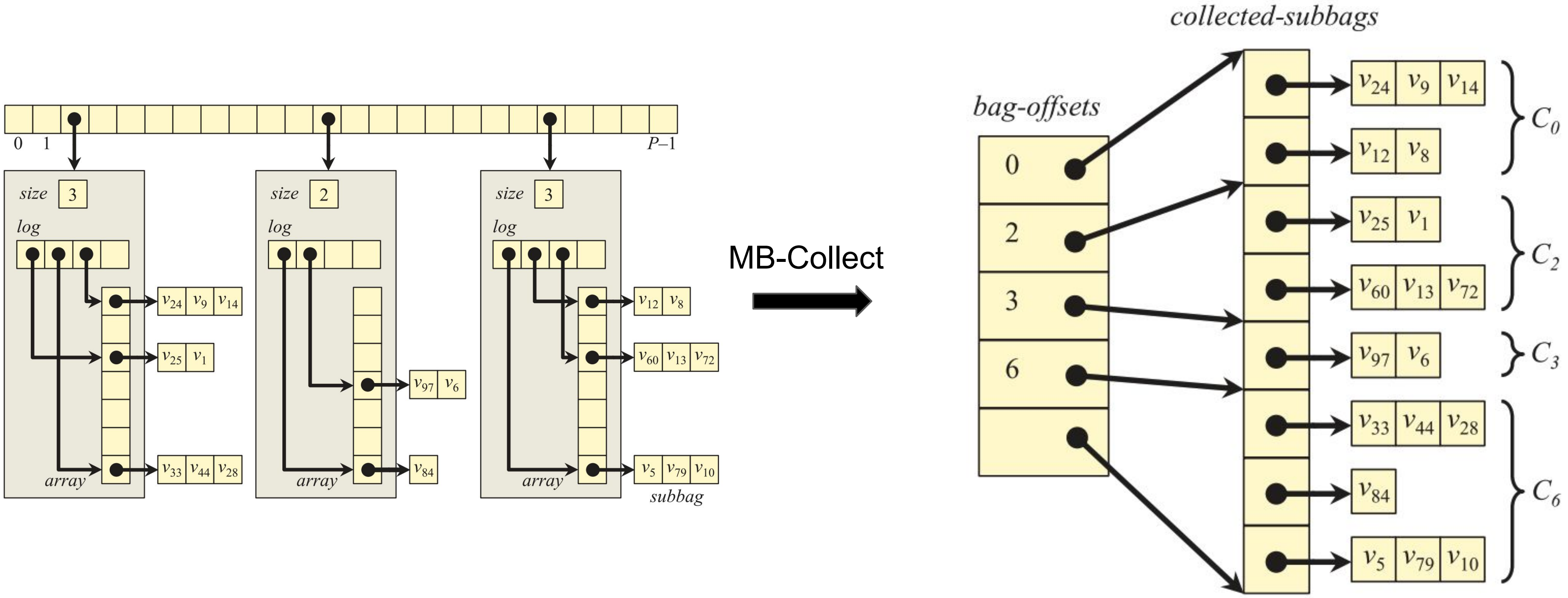
```
MB-INSERT(Q, v, k)
p = GET-WORKER-ID()
if Q[p].array[k] == NIL
    APPEND(Q[p].log, k)
    Q[p].array[k] = new subbag
APPEND(Q[p].array[k], v)
```



MB-Collect (Board)



MB-Collect (Board)



Multibags Performance

- MB-insert is Theta(1)
- For m number of total subbags
- MB-Collect is O(m + Color + P) work and O(log(m) + Color + log(P)) span
- For all the obvious reasons… (use prefix sum and friends)

Final Theoretical Result

- Suppose that Prism colors a degree-D data graph $G=(V, E)$ using xi colors and then executes the data-graph computation (G, f, Q_0). Then, on P processors, Prism executes updates on all vertices in the activation set Q_r for a round r using

  Work: $O(\mathrm{size}(Q_r) + P)$

  Span: $O(\xi * (\log(Q_r/\xi) + \log(D)) + \log(P))$,

  where $\mathrm{size}(Q) := |Q| + \sum_{v \in Q} \deg(v)$

PRISM(G, f, Q_0)

```
χ = COLOR-GRAPH(G)
r = 0
Q = Q_0
while Q ≠ ∅
    𝒞 = MB-COLLECT(Q)
    for C ∈ 𝒞
        parallel for v ∈ C
            active[v] = FALSE
            S = f(v)
            parallel for u ∈ S
                if CAS(active[u], FALSE, TRUE)
                    MB-INSERT(Q, u, color[u])
        r = r + 1
```

Work: Theta(deg(v)); Span: Theta(log(deg(v)))

Work: Theta(S); Span: Theta(log(S))

```
PRISM(G, f, Q_0)
χ = COLOR-GRAPH(G)
r = 0
Q = Q_0
while Q ≠ ∅
    𝒞 = MB-COLLECT(Q)
    for C ∈ 𝒞
        parallel for v ∈ C
            active[v] = FALSE
            S = f(v)
            parallel for u ∈ S
                if CAS(active[u], FALSE, TRUE)
                    MB-INSERT(Q, u, color[u])
        r = r + 1
```

Work: Theta(size(C));

Span: Theta(log(C) + log(D))

```
PRISM(G, f, Q_0)
χ = COLOR-GRAPH(G)
r = 0
Q = Q_0
while Q ≠ ∅
    C = MB-COLLECT(Q)
    for C ∈ C
        parallel for v ∈ C
            active[v] = FALSE
            S = f(v)
            parallel for u ∈ S
                if CAS(active[u], FALSE, TRUE)
                    MB-INSERT(Q, u, color[u])
    r = r + 1
```

Work: Theta(size(Q_r) + xi)

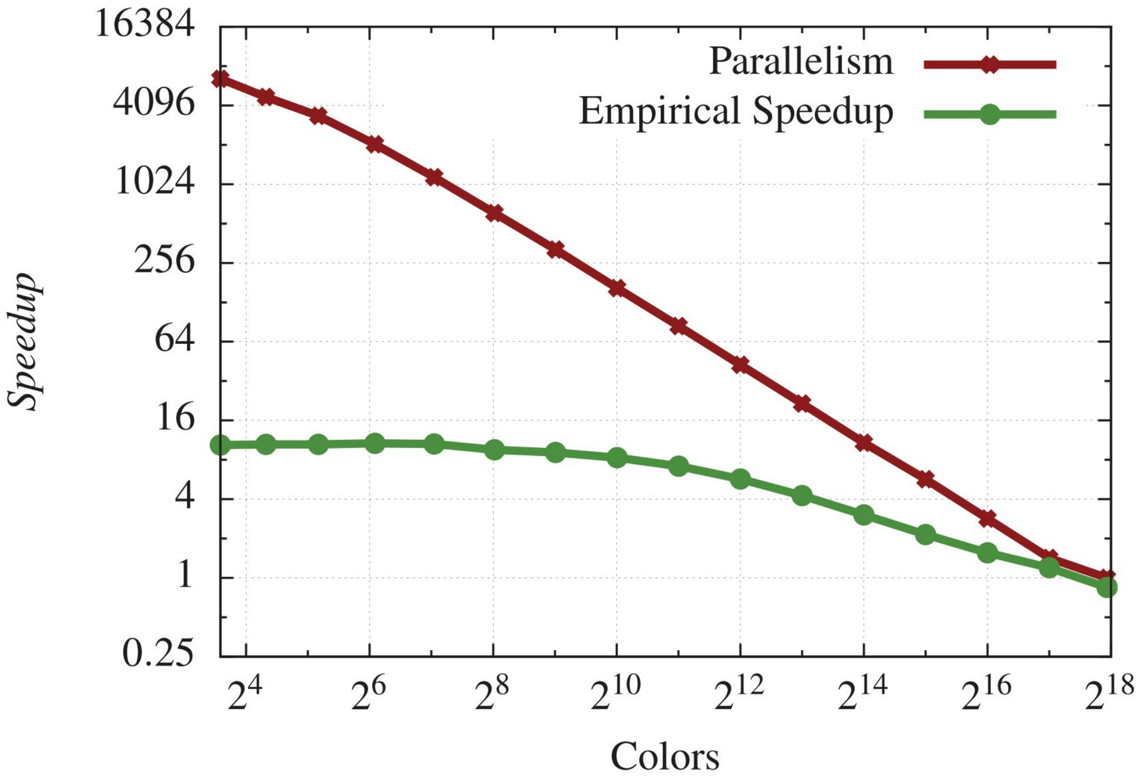
Span: Theta(xi*(log(Q_r/xi) + log(D)))

```
PRISM(G, f, Q0)
χ = COLOR-GRAPH(G)
r = 0
Q = Q0
while Q ≠ ∅
    𝒞 = MB-COLLECT(Q)    O(m + Color + P) work and O(log(m) + Color + log(P)) span
    for C ∈ 𝒞
        parallel for v ∈ C
            active[v] = FALSE
            S = f(v)
            parallel for u ∈ S
                if CAS(active[u], FALSE, TRUE)
                    MB-INSERT(Q, u, color[u])
        r = r + 1
```

Experimental Results

Graph	$\|V\|$	$\|E\|$	χ	CILK+LOCKS	PRISM	PRISM-R	*Coloring*
cage15	5,154,860	94,044,700	17	36.9	35.5	35.6	12%
liveJournal	4,847,570	68,475,400	333	36.8	21.7	22.3	12%
randLocalDim25	1,000,000	49,992,400	36	26.7	14.4	14.6	18%
randLocalDim4	1,000,000	41,817,000	47	19.5	12.5	13.7	14%
rmat2Million	2,097,120	39,912,600	72	22.5	16.6	16.8	12%
powerGraph2M	2,000,000	29,108,100	15	12.1	9.8	10.1	13%
3dgrid5m	5,000,210	15,000,600	6	10.3	10.3	10.4	7%
2dgrid5m	4,999,700	9,999,390	4	17.7	8.9	9.0	4%
web-Google	916,428	5,105,040	43	3.9	2.4	2.4	8%
web-BerkStan	685,231	7,600,600	200	3.9	2.4	2.7	8%
web-Stanford	281,904	2,312,500	62	1.9	0.9	1.0	11%
web-NotreDame	325,729	1,469,680	154	1.1	0.8	0.8	12%

Coloring Need Not Be Perfect



Thanks! Questions?