6.172 Performance Engineering of Software Systems

Lecture 13: Chromatic Scheduling

Tao B. Schardl

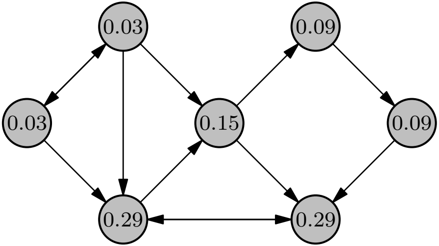
MIT Computer Science and Artificial Intelligence Laboratory

October 23, 2012

PageRank

Definition

Given a graph $(P, L(P))$ of pages P and links between pages $L(P)$, the ***PageRank*** $PR(p_i)$ of a page p_i is the probability that a person who randomly follows links will stop at page p_i.

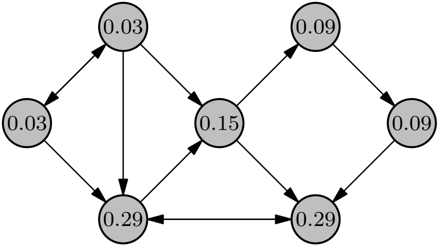


PageRank

Formally, the PageRank $PR(p_i)$ of page p_i is defined by

$$PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$$

- $N(p_i)$ is the set of pages that link to p_i,
- $L(q)$ is the set of outgoing links from page q, and
- d is the probability of following any link on a page.

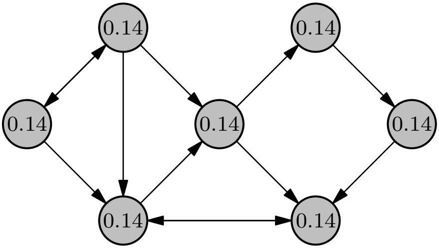


PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



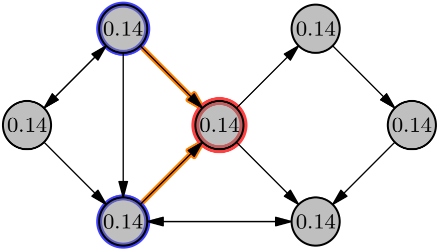
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



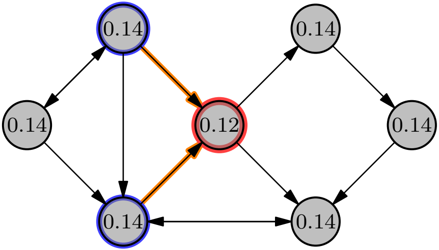
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



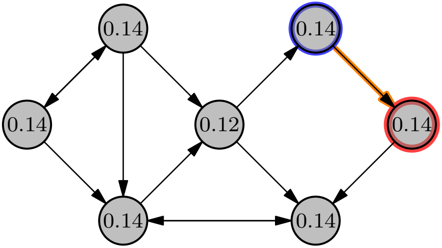
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



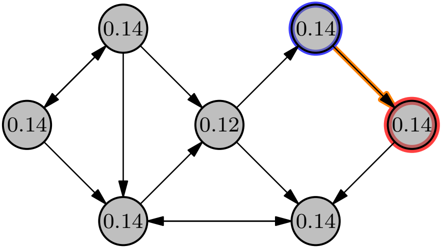
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



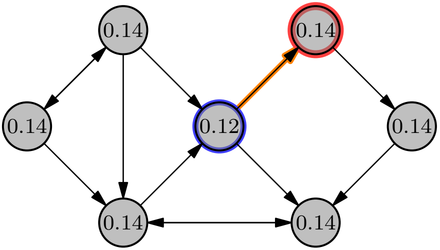
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



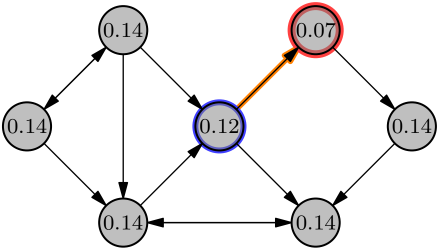
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



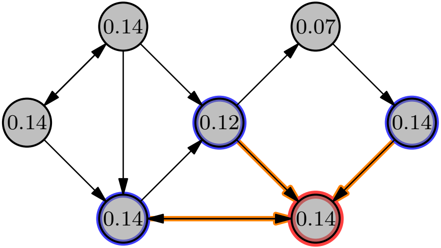
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



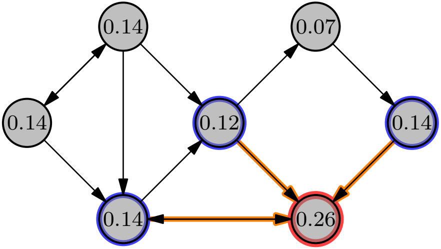
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



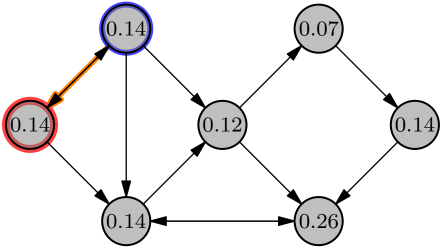
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



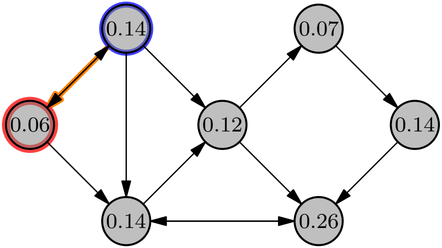
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.

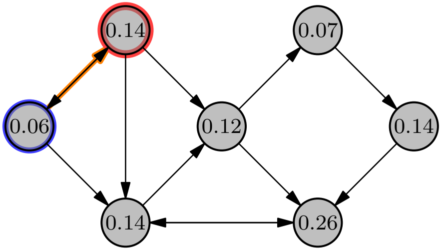


- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



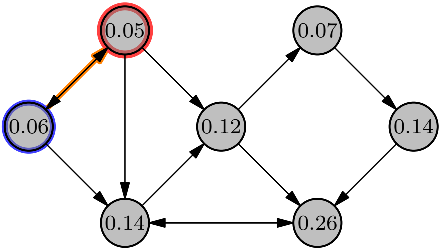
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



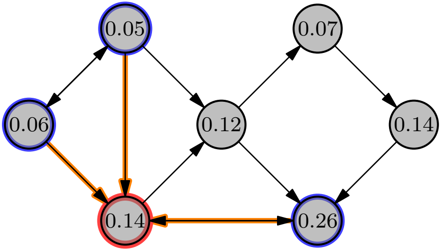
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



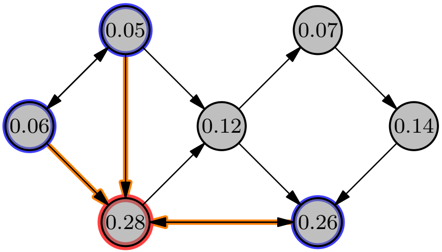
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



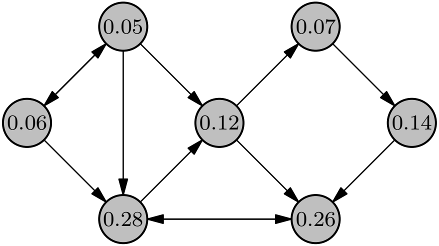
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.

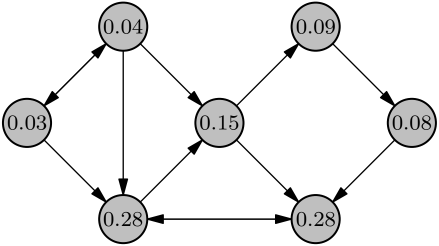


- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



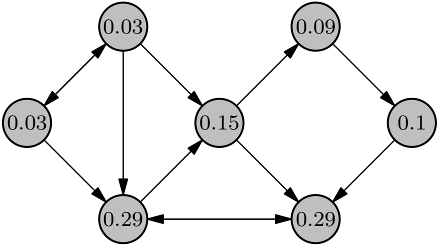
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



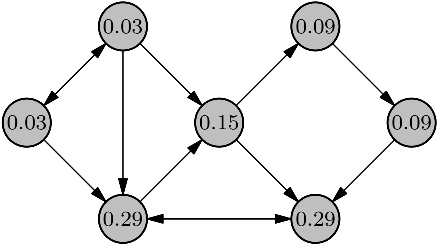
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



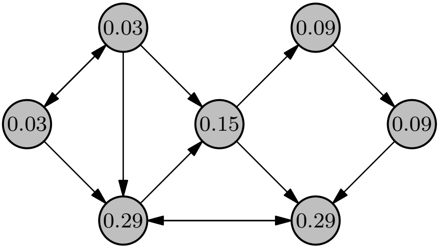
- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

PageRank

Problem

Given a graph $(P, L(P))$ of pages P connected by links $L(P)$, compute the PageRank of each page in P.

Main Idea: Compute PageRanks iteratively until convergence.



- Initially, all PageRanks are $\frac{1}{|P|}$.
- Update using $PR(p_i) = \frac{1-d}{|P|} + d \sum_{q \in N(p_i)} \frac{PR(q)}{|L(q)|}$
- Example uses $d = 0.85$.

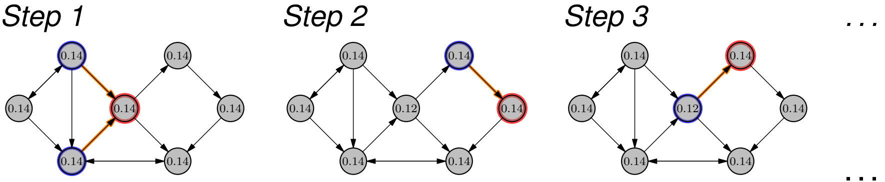
PageRank

```
bool done = false;
while (!done) {  // Iterate until convergence
  done = true;
  for (int p = 0; p < N; ++p) {  // Scan pages
    // Accumulate weighted PageRanks of neighbors
    double sum = 0;
    for (int l = inEdgeList[p]; l < inEdgeList[p+1]; ++l) {
      int q = inEdges[l];
      sum += pageRank[q] / (outEdgeList[q+1] - outEdgeList[q]);
    }
    // Compute the new PageRank for p
    double newPageRank = (1-d) / N + d * sum;
    // If change to PageRank exceeds tolerance,
    // update PageRank and ensure we reiterate.
    if (abs(newPageRank - pageRank[p]) > tolerance) {
      pageRank[p] = newPageRank;
      done = false;
    }
  }
}
```

Parallel PageRank

Problem

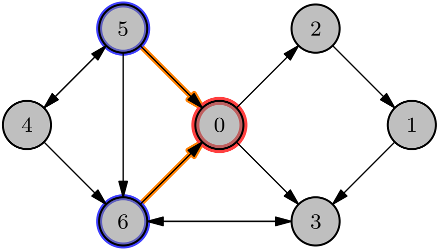
How do we update PageRanks in parallel?



Problem

How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?

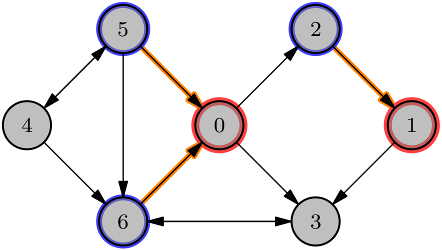


Parallel PageRank

Problem

How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?

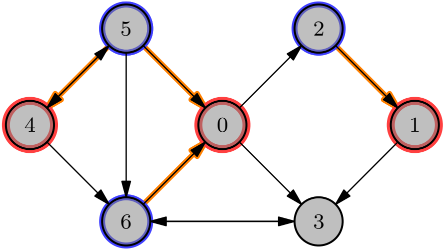


Parallel PageRank

Problem

How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?

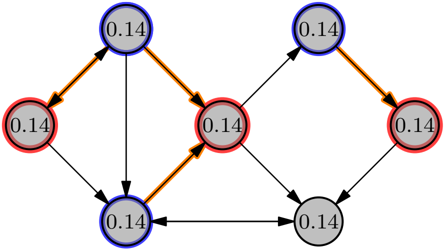


Parallel PageRank

Problem

How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?

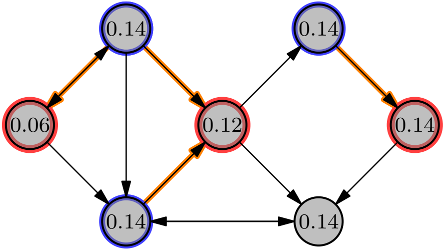


Parallel PageRank

Problem

How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?

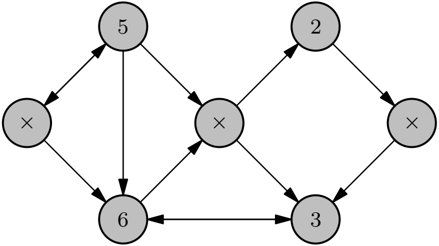


Parallel PageRank

Problem

How do we update PageRanks in parallel?

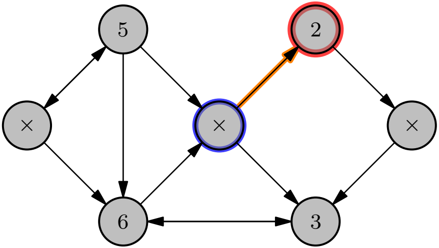
Consider: What do we do to update a page's PageRank?



Problem

How do we update PageRanks in parallel?

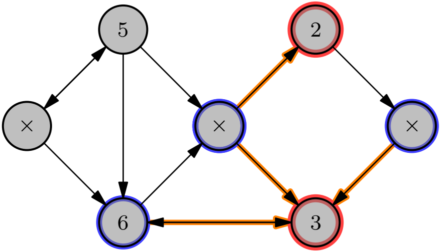
Consider: What do we do to update a page's PageRank?



Problem

How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?

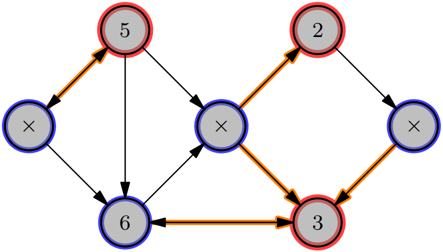


Parallel PageRank

Problem

How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?

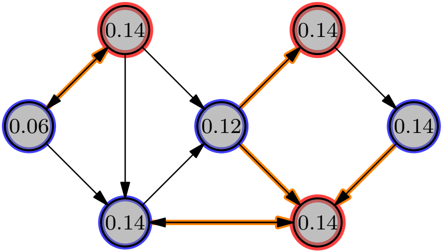


Parallel PageRank

Problem

How do we update PageRanks in parallel?

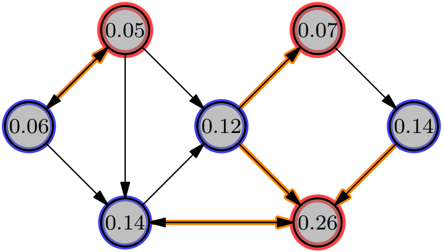
Consider: What do we do to update a page's PageRank?



Problem

How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?

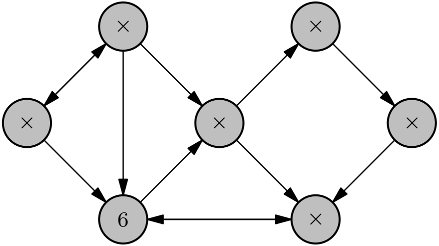


Parallel PageRank

Problem

How do we update PageRanks in parallel?

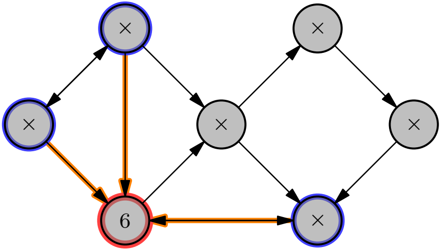
Consider: What do we do to update a page's PageRank?



Problem

How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?

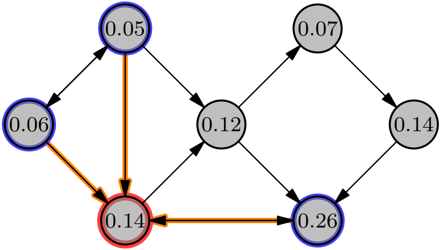


Parallel PageRank

Problem

How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?

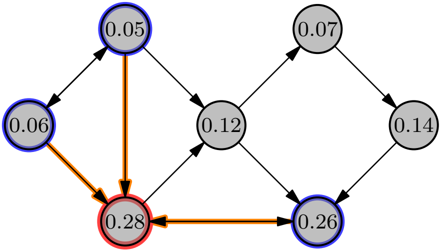


Parallel PageRank

Problem

How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?

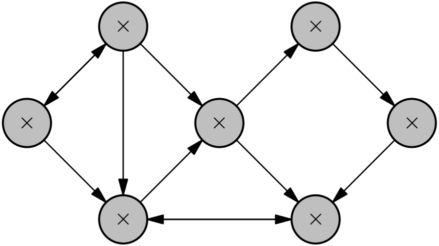


Parallel PageRank

Problem

How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?

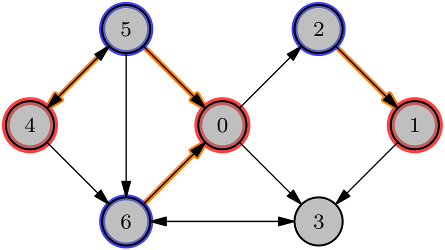


Parallel PageRank

Problem

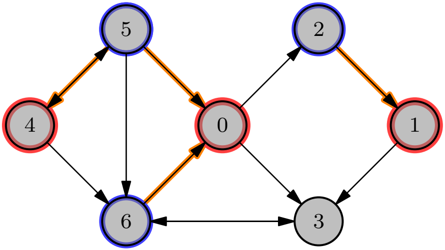
How do we update PageRanks in parallel?

Consider: What do we do to update a page's PageRank?



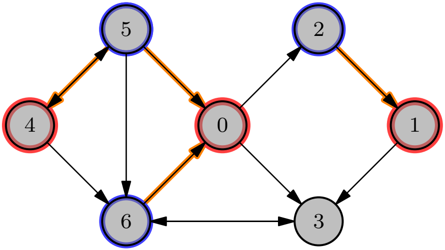
Insight: Two pages that are not directly linked can be updated in parallel.

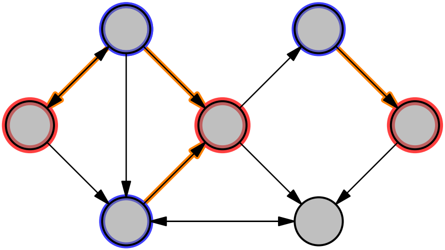
Consider the sets of pages that can be updated in parallel.



Parallel PageRank

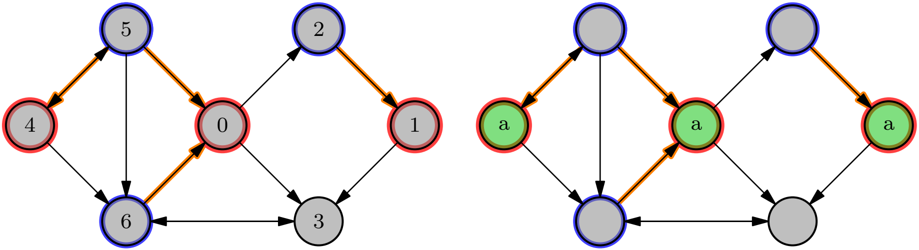
Consider the sets of pages that can be updated in parallel.



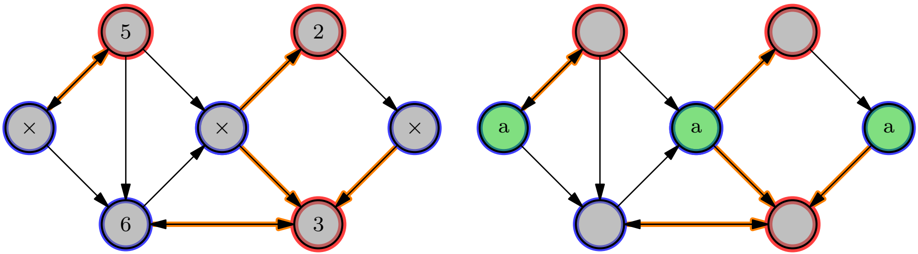


Parallel PageRank

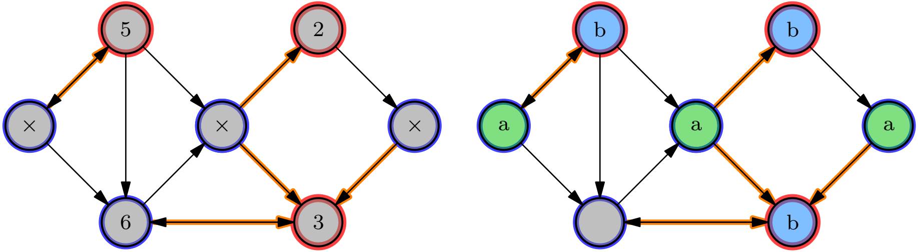
Consider the sets of pages that can be updated in parallel.



Consider the sets of pages that can be updated in parallel.

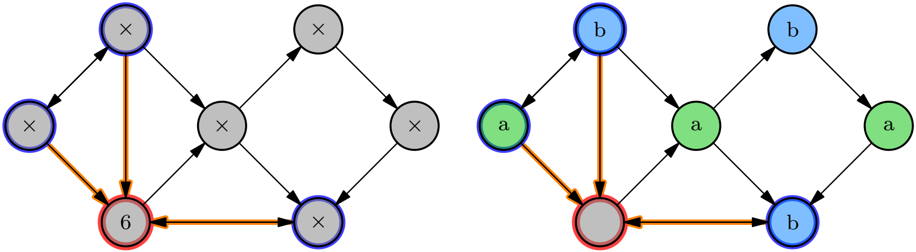


Consider the sets of pages that can be updated in parallel.

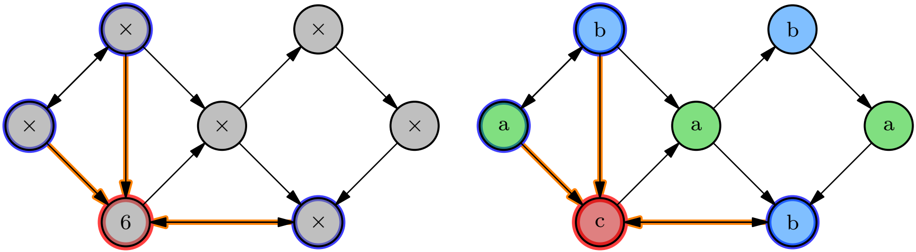


Parallel PageRank

Consider the sets of pages that can be updated in parallel.

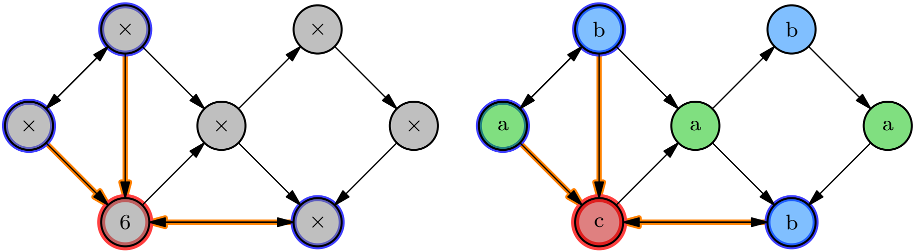


Consider the sets of pages that can be updated in parallel.



Parallel PageRank

Consider the sets of pages that can be updated in parallel.



These sets define a ***coloring*** of the (undirected) graph — an assignment of labels, or ***colors***, to the vertices of the graph such that no two adjacent vertices have the same color.

Outline

1. Chromatic Scheduling
2. Parallel PageRank
3. The Bag Data Structure
4. Simulating fluid flows

Outline

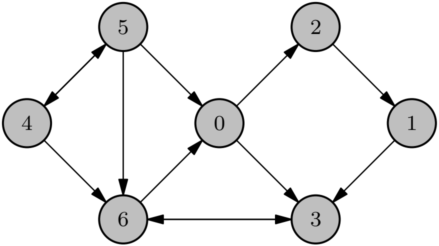
Coloring data graphs

Problem

Given a graph $G = (V, E)$, perform iterative updates on the vertices and edges of the graph in parallel while avoiding races.

Solution

Color the conflict graph, then process vertices of the same color in parallel.



Two vertices u and v are connected in the ***conflict graph*** if processing u reads or writes memory written by processing v.

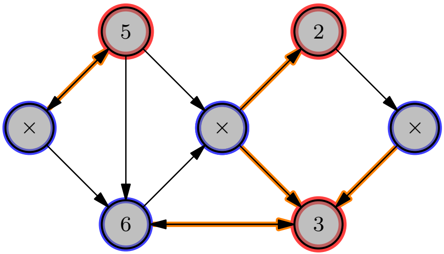
Coloring data graphs

Problem

Given a graph $G = (V, E)$, perform iterative updates on the vertices and edges of the graph in parallel while avoiding races.

Solution

Color the conflict graph, then process vertices of the same color in parallel.



Two vertices u and v are connected in the ***conflict graph*** if processing u reads or writes memory written by processing v.

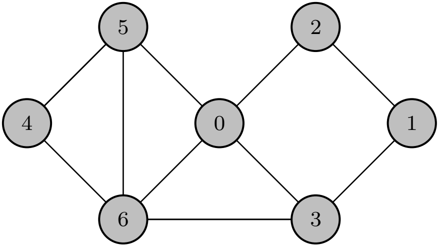
Coloring data graphs

Problem

Given a graph $G = (V, E)$, perform iterative updates on the vertices and edges of the graph in parallel while avoiding races.

Solution

Color the conflict graph, then process vertices of the same color in parallel.



Two vertices u and v are connected in the ***conflict graph*** if processing u reads or writes memory written by processing v.

For PageRank: the conflict graph is the undirected input graph $(P, L(P))$.

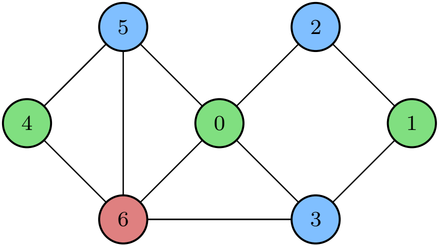
Coloring data graphs

Problem

Given a graph $G = (V, E)$, perform iterative updates on the vertices and edges of the graph in parallel while avoiding races.

Solution

Color the conflict graph, then process vertices of the same color in parallel.

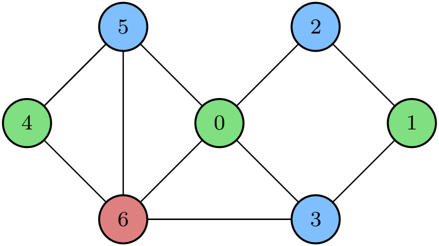


Two vertices u and v are connected in the ***conflict graph*** if processing u reads or writes memory written by processing v.

For PageRank: the conflict graph is the undirected input graph $(P, L(P))$.

Why does coloring work?

An ***independent set*** is a set of vertices such that no two vertices in the set are adjacent.



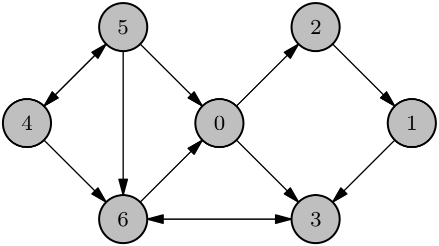
- The vertices in an independent set of the conflict graph are not adjacent.
- Processing these vertices therefore does not cause a race.
- Coloring the conflict graph ensures that no two connected nodes share the same color.
- By coloring the conflict graph, each set of nodes of the same color is an independent set.

Outline

1. Chromatic Scheduling
2. Parallel PageRank
3. The Bag Data Structure
4. Simulating fluid flows

Back to parallel PageRank

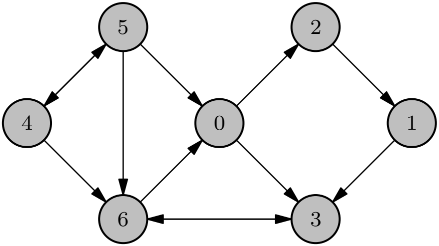
Let's summarize our parallel PageRank algorithm:



Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

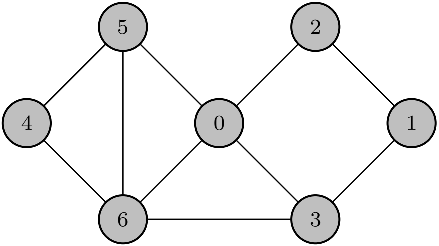
1. Color the conflict graph.



Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

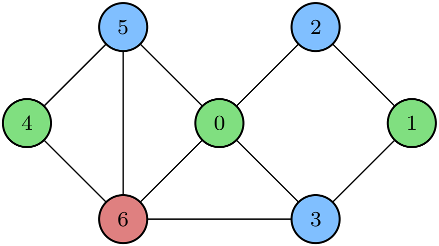
1. Color the conflict graph.



Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

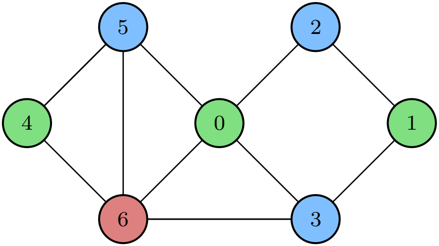
1. Color the conflict graph.



Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

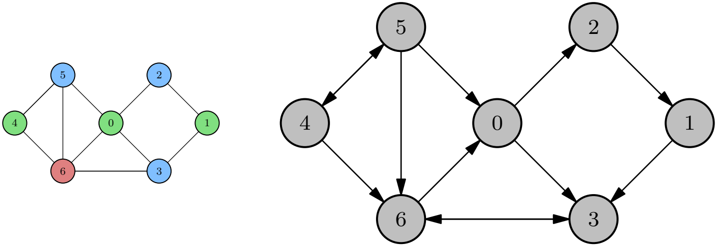
1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.



Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

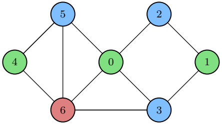
1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.

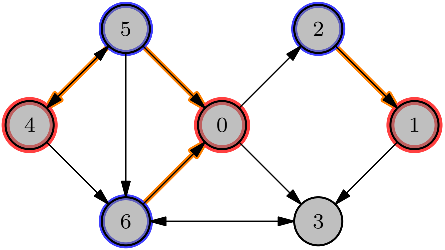


Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.

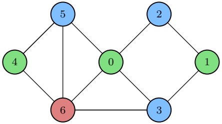


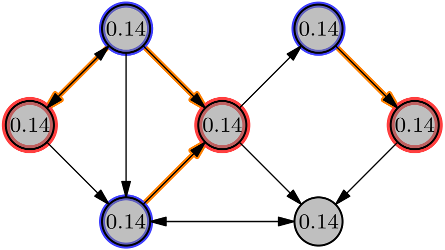


Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.

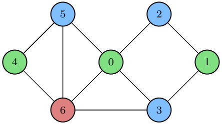


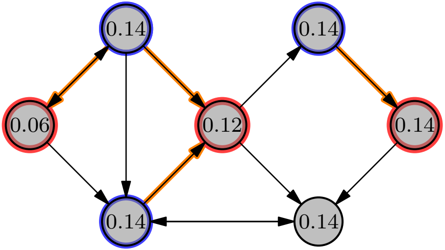


Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.

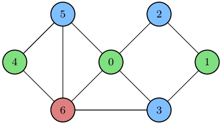


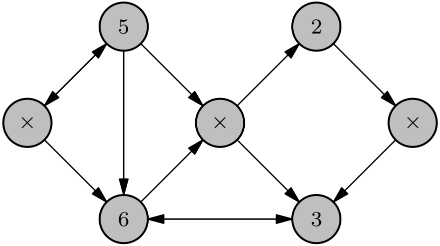


Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.

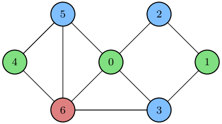


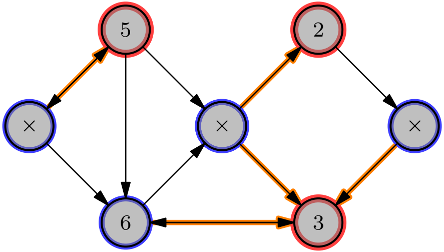


Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.

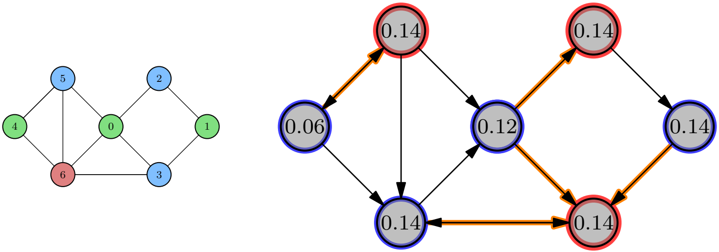




Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

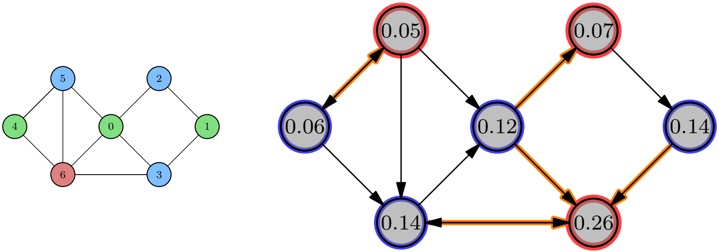
1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.



Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

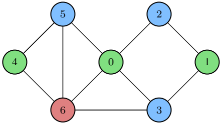
1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.

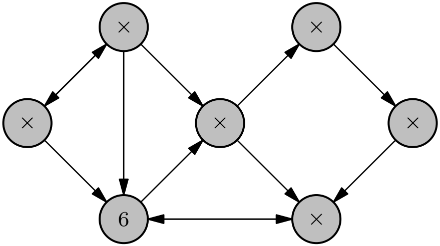


Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.

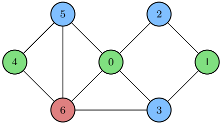


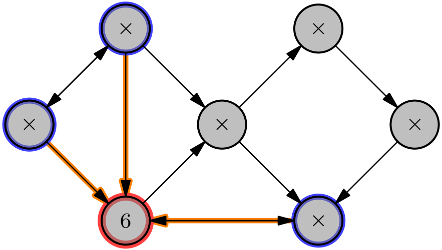


Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

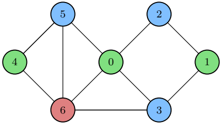
1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.

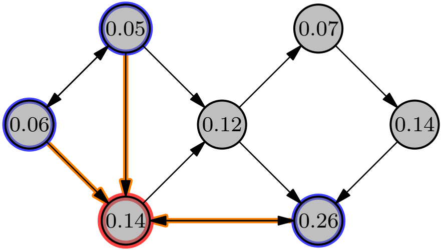




Let's summarize our parallel PageRank algorithm:

1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.

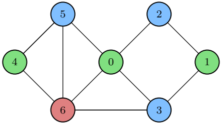


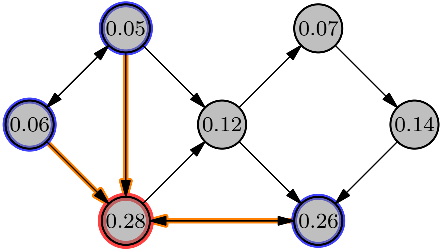


Back to parallel PageRank

Let's summarize our parallel PageRank algorithm:

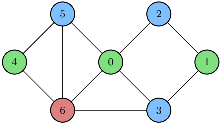
1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.

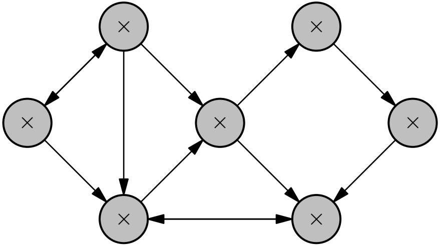




Let's summarize our parallel PageRank algorithm:

1. Color the conflict graph.
2. Process the colors serially, but process vertices of the same color in parallel.





Back to parallel PageRank

```
int numColors = colorGraph();  // Color the graph
bool done = false;
while (!done) {  // Iterate until convergence
  cilk::reducer< cilk::opand<bool> > done_r();

  // Process colors serially
  for (int c = 0; c < numColors; ++c) {
    // Process pages of same color in parallel
    cilk_for (int i = 0; i < numColoredPages[c]; ++i) {
      int p = coloredPages[c][i];
      int newPageRank = computePageRank(p);
      if (abs(newPageRank - pageRank[p]) > tolerance) {
        pageRank[p] = newPageRank;
        *done_r &= false;
  } } }
  done_r.move_out(done);
}
```

Back to parallel PageRank

```
int numColors = colorGraph();  // Color the graph
bool done = false;
while (!done) {  // Iterate until convergence
  cilk::reducer< cilk::opand<bool> > done_r();

  // Process colors serially
  for (int c = 0; c < numColors; ++c) {
    // Process pages of same color in parallel
    cilk_for (int i = 0; i < numColoredPages[c]; ++i) {
      int p = coloredPages[c][i];
      int newPageRank = computePageRank(p);
      if (abs(newPageRank - pageRank[p]) > tolerance) {
        pageRank[p] = newPageRank;
        *done_r &= false;
  } } }
  done_r.move_out(done);
}
```

Question: Why don't we need to recolor the graph each iteration?

Back to parallel PageRank

```
int numColors = colorGraph();  // Color the graph
bool done = false;
while (!done) {  // Iterate until convergence
  cilk::reducer< cilk::opand<bool> > done_r();

  // Process colors serially
  for (int c = 0; c < numColors; ++c) {
    // Process pages of same color in parallel
    cilk_for (int i = 0; i < numColoredPages[c]; ++i) {
      int p = coloredPages[c][i];
      int newPageRank = computePageRank(p);
      if (abs(newPageRank - pageRank[p]) > tolerance) {
        pageRank[p] = newPageRank;
        *done_r &= false;
  } } }
  done_r.move_out(done);
}
```

Question: Why don't we need to recolor the graph each iteration?
Answer: The graph is static, so the same coloring always works.

Performance of parallel PageRank

What's the theoretical performance of this parallel PageRank?

To update all PageRanks in a graph $(P, L(P))$ in a single iteration:
Work:
Span:

Performance of parallel PageRank

What's the theoretical performance of this parallel PageRank?

To update all PageRanks in a graph $(P, L(P))$ in a single iteration:
Work: $W = \Theta(P + L(P))$
Span:

Performance of parallel PageRank

What's the theoretical performance of this parallel PageRank?

To update all PageRanks in a graph $(P, L(P))$ in a single iteration:
Work: $W = \Theta(P + L(P))$
Span: $S = \langle\text{number of colors}\rangle \cdot \langle\text{span to process one color}\rangle$

Performance of parallel PageRank

What's the theoretical performance of this parallel PageRank?

To update all PageRanks in a graph $(P, L(P))$ in a single iteration:
Work: $W = \Theta(P + L(P))$
Span: $S = \langle\text{number of colors}\rangle \cdot \langle\text{span to process one color}\rangle$

We can process all pages P_{a} of color a in span $O(\lg P_{\text{a}})$.

Performance of parallel PageRank

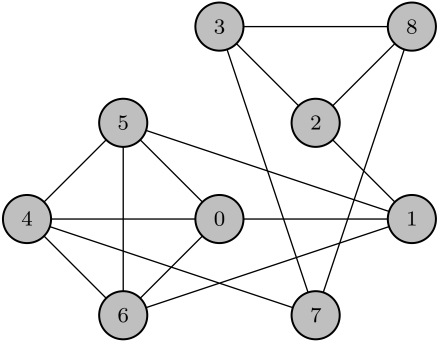
What's the theoretical performance of this parallel PageRank?

To update all PageRanks in a graph $(P, L(P))$ in a single iteration:
Work: $W = \Theta(P + L(P))$
Span: $S = \langle \text{number of colors} \rangle \cdot \langle \text{span to process one color} \rangle$

We can process all pages P_{a} of color a in span $O(\lg P_{\text{a}})$.
Question: How many colors do we need?

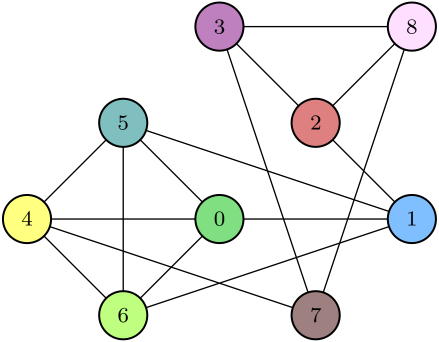
Number of colors

Question: How many colors are needed to color a graph $G = (V, E)$?



Number of colors

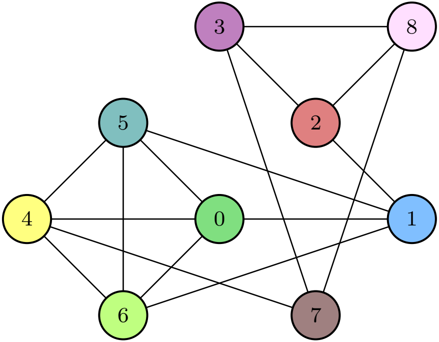
Question: How many colors are needed to color a graph $G = (V, E)$?



- Give each vertex its own color.

Number of colors

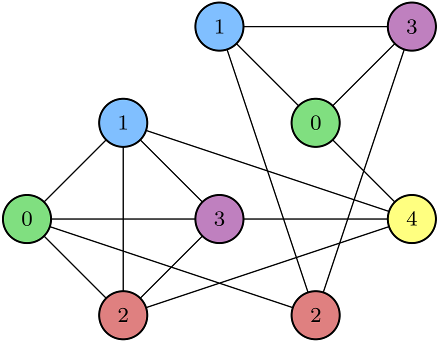
Question: How many colors are needed to color a graph $G = (V, E)$?



- Give each vertex its own color.
  - Uses $|V|$ colors in total.
  - Equivalent to processing the graph serially.

Number of colors

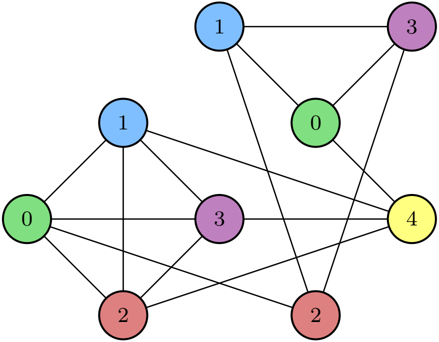
Question: How many colors are needed to color a graph $G = (V, E)$?



- Give each vertex its own color.
  - Uses $|V|$ colors in total.
  - Equivalent to processing the graph serially.
- If Δ is the maximum degree of any vertex in V, we can color G using $\Delta + 1$ colors.

Number of colors

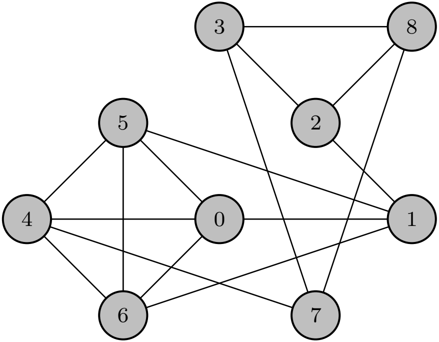
Question: How many colors are needed to color a graph $G = (V, E)$?



- Give each vertex its own color.
  - Uses $|V|$ colors in total.
  - Equivalent to processing the graph serially.
- If Δ is the maximum degree of any vertex in V, we can color G using $\Delta + 1$ colors.
- Finding the minimum coloring of a general graph is NP-complete, but we don't necessarily need a minimum coloring.

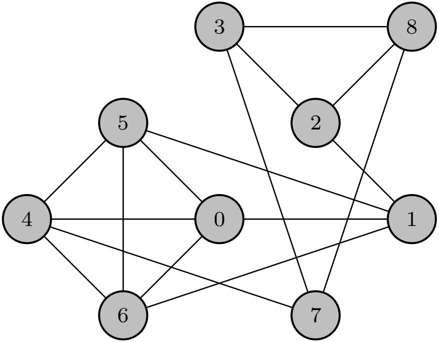
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?



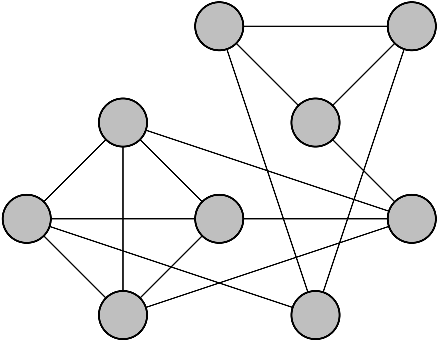
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



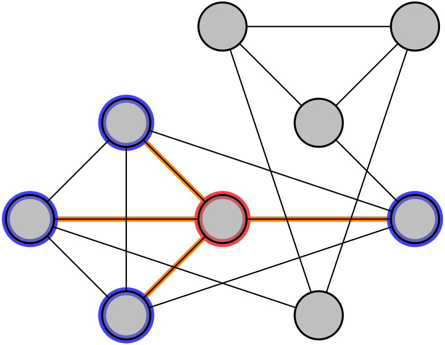
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



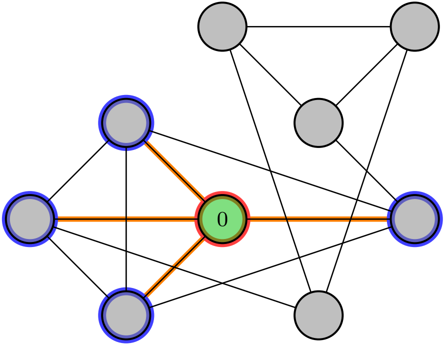
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



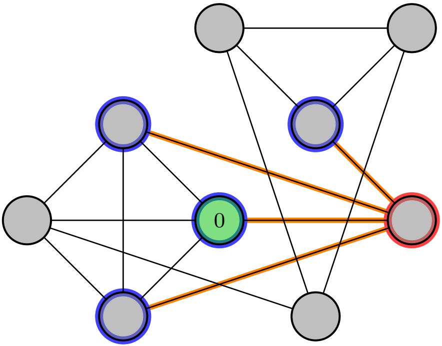
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



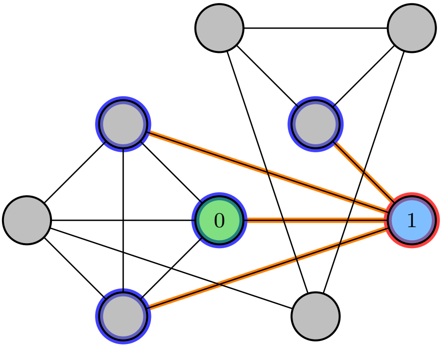
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



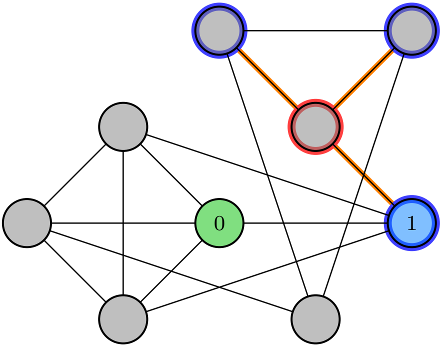
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



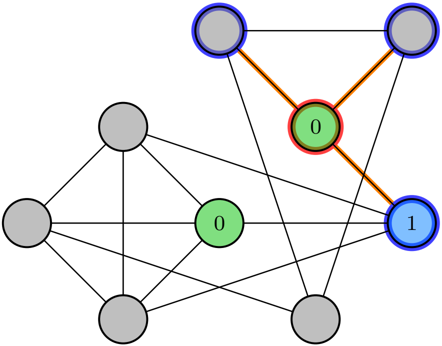
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



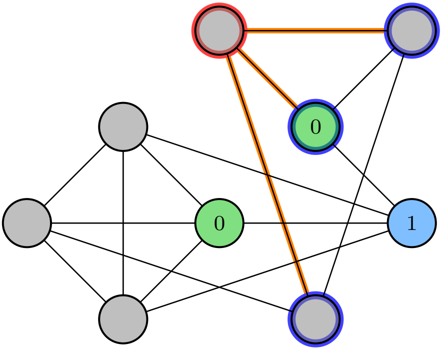
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



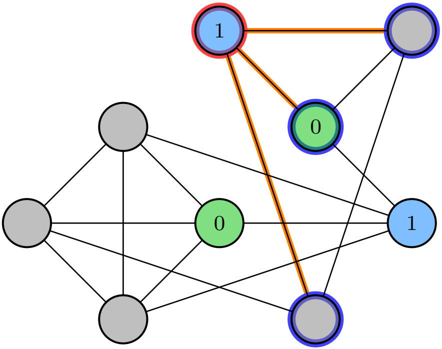
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



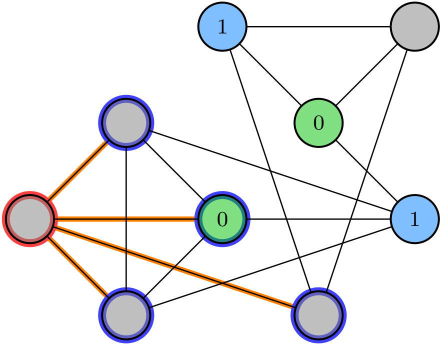
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



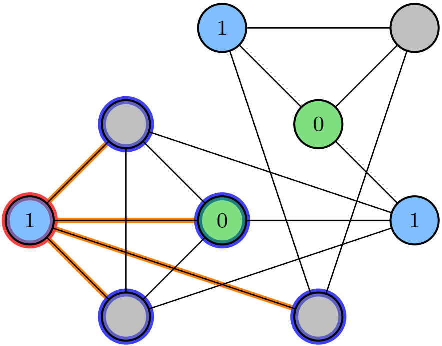
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



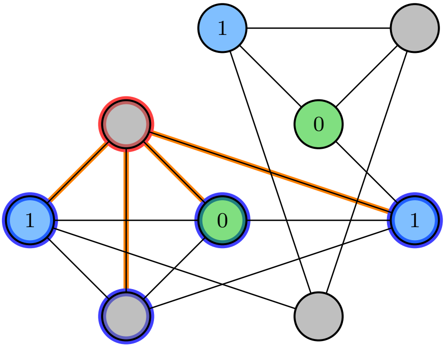
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



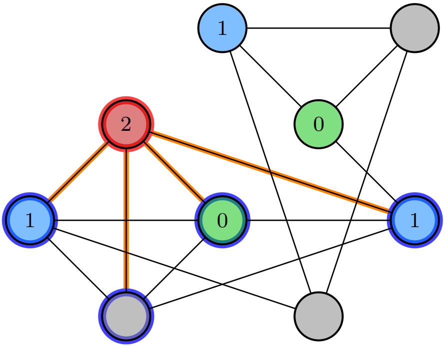
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



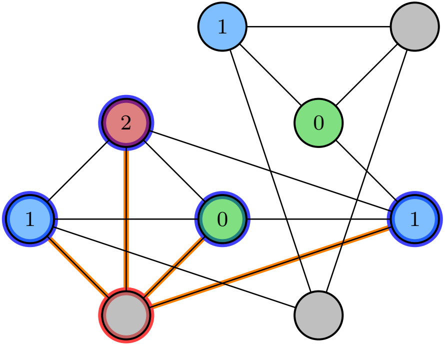
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



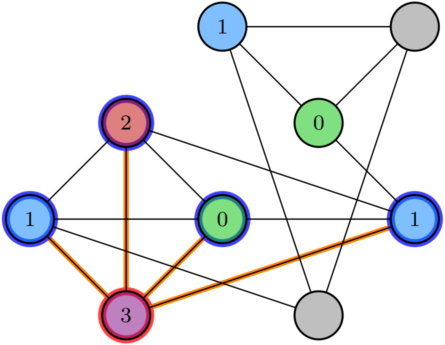
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



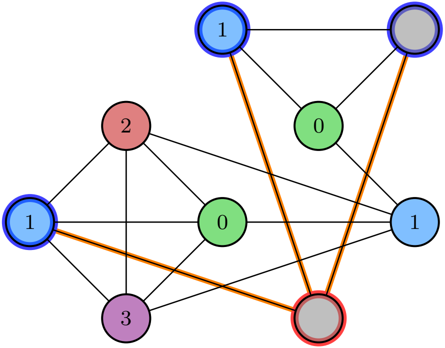
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



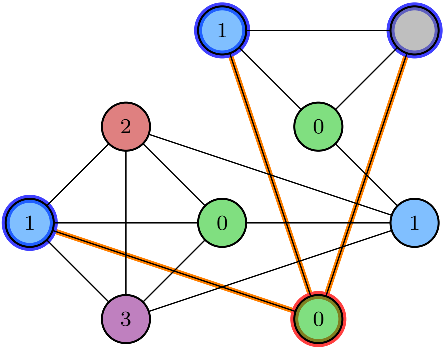
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



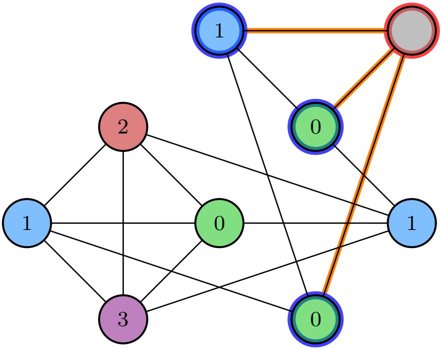
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



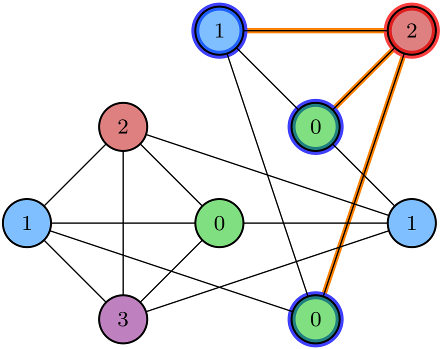
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



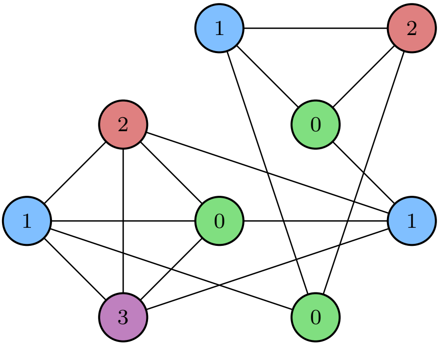
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



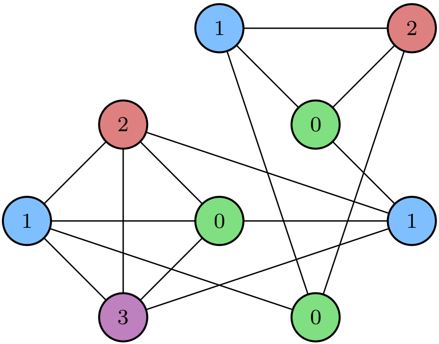
Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



Serial coloring algorithm

Question: How do we find a $\Delta + 1$ coloring of a graph G (serially)?
Answer: Greedily pick the smallest available color for each node.



This algorithm is guaranteed to find a $\Delta + 1$ coloring, although it may do better.

Serial coloring algorithm

```
char usedColors[maxDegree];
memset(usedColors, 0, maxDegree);
for (int i = 0; i < N; ++i) {  // Scan vertices
  int degree = nodes[i+1] - nodes[i];
  // Tally colors of neighbors
  for (int j = nodes[i]; j < nodes[i+1]; ++j) {
    if (colors[edges[j]] < degree)
      usedColors[colors[edges[j]]] = 1;
  }
  int color;
  for (color = 0; color < degree; ++color) {
    if (usedColors[color] == 0)
      break;
  }
  colors[i] = color;
  memset(usedColors, 0, degree);
}
```

Serial coloring algorithm

```
char usedColors[maxDegree];
memset(usedColors, 0, maxDegree);
for (int i = 0; i < N; ++i) {  // Scan vertices
  int degree = nodes[i+1] - nodes[i];
  // Tally colors of neighbors
  for (int j = nodes[i]; j < nodes[i+1]; ++j) {
    if (colors[edges[j]] < degree)
      usedColors[colors[edges[j]]] = 1;
  }
  int color;
  for (color = 0; color < degree; ++color) {
    if (usedColors[color] == 0)
      break;
  }
  colors[i] = color;
  memset(usedColors, 0, degree);
}
```

Work:

Serial coloring algorithm

```
char usedColors[maxDegree];
memset(usedColors, 0, maxDegree);
for (int i = 0; i < N; ++i) {  // Scan vertices
  int degree = nodes[i+1] - nodes[i];
  // Tally colors of neighbors
  for (int j = nodes[i]; j < nodes[i+1]; ++j) {
    if (colors[edges[j]] < degree)
      usedColors[colors[edges[j]]] = 1;
  }
  int color;
  for (color = 0; color < degree; ++color) {
    if (usedColors[color] == 0)
      break;
  }
  colors[i] = color;
  memset(usedColors, 0, degree);
}
```

Work:
$W = \Theta(V + E)$

Serial coloring algorithm

```
char usedColors[maxDegree];
memset(usedColors, 0, maxDegree);
for (int i = 0; i < N; ++i) {  // Scan vertices
  int degree = nodes[i+1] - nodes[i];
  // Tally colors of neighbors
  for (int j = nodes[i]; j < nodes[i+1]; ++j) {
    if (colors[edges[j]] < degree)
      usedColors[colors[edges[j]]] = 1;
  }
  int color;
  for (color = 0; color < degree; ++color) {
    if (usedColors[color] == 0)
      break;
  }
  colors[i] = color;
  memset(usedColors, 0, degree);
}
```

Work:
$W = \Theta(V + E)$
Work-efficient parallel coloring algorithms are possible.

Performance of parallel PageRank

Theoretical performance:

To update all PageRanks in a graph $(P, L(P))$ in a single iteration:
Work: $W = \Theta(P + L(P))$
Span:
$S = \langle\text{number of colors}\rangle \cdot \langle\text{span to process one color}\rangle = O(\Delta \lg P/\Delta)$

Advantages of chromatic scheduling

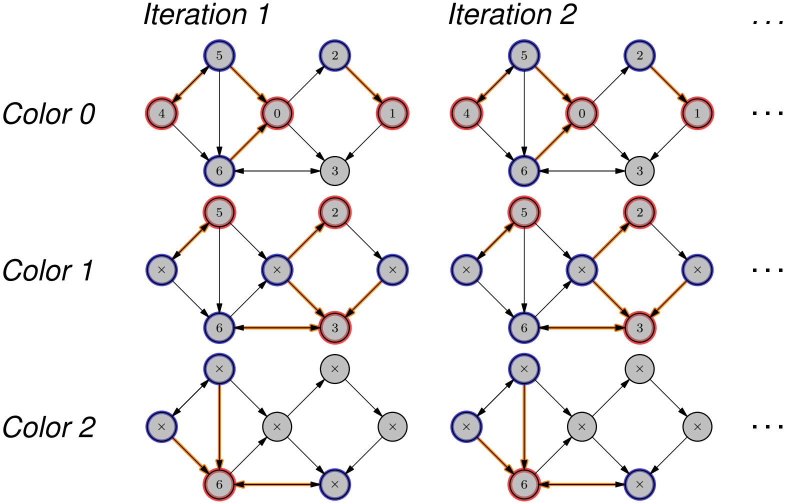
Chromatic scheduling offers many nice properties.

- For a static graph, the same coloring always works. Computing a chromatic schedule can be done as *precomputation*.
  - Coloring is *relatively cheap* when the work and span of the main computation exceeds the work of the work-efficient serial coloring algorithm.
  - Work-efficient parallel coloring algorithms are also possible.
- Processing the colors in the same order every time processes the graph *deterministically*.
- Chromatic scheduling handles many problems that can be viewed as performing local updates to vertices and edges in a graph, including Loopy belief propagation, Gibbs sampling, fluid dynamics simulation, and many machine-learning algorithms.

Outline

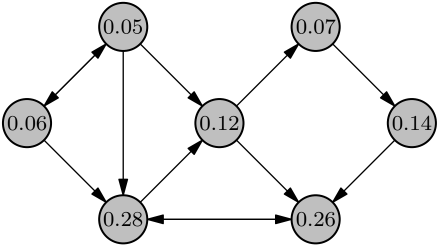
Optimizing parallel PageRank

Question: Can we improve our parallel PageRank code?



Question: Can we improve our parallel PageRank code?
Consider: Which PageRanks change significantly (by more than the threshold) after the first iteration?

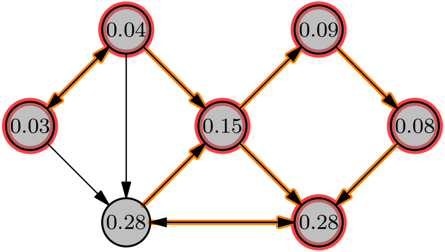
After Iteration 1:



Optimizing parallel PageRank

Question: Can we improve our parallel PageRank code?
Consider: Which PageRanks change significantly (by more than the threshold) after the first iteration?

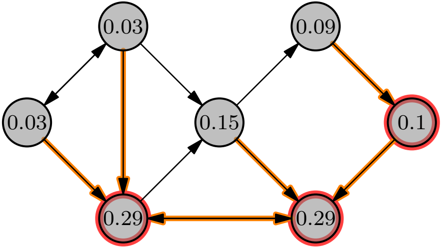
Iteration 2 significant updates:



Optimizing parallel PageRank

Question: Can we improve our parallel PageRank code?
Consider: Which PageRanks change significantly (by more than the threshold) after the first iteration?

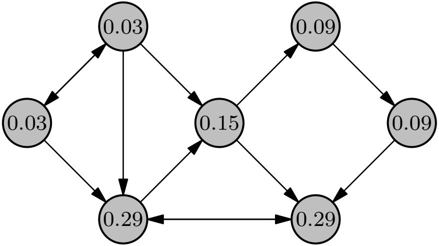
Iteration 3 significant updates:



Optimizing parallel PageRank

Question: Can we improve our parallel PageRank code?
Consider: Which PageRanks change significantly (by more than the threshold) after the first iteration?

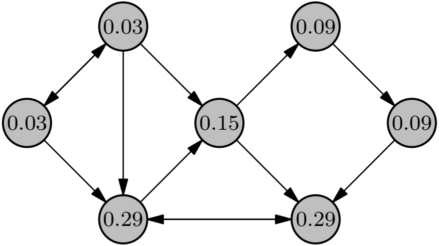
Iteration 4 significant updates:



Optimizing parallel PageRank

Question: Can we improve our parallel PageRank code?
Consider: Which PageRanks change significantly (by more than the threshold) after the first iteration?

Iteration 4 significant updates:



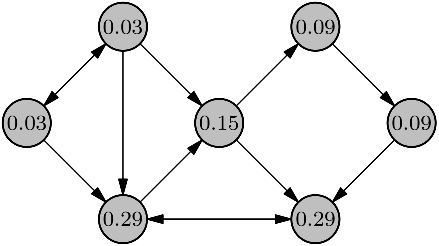
Idea: If a page's PageRank converges, don't reprocess it immediately.

- Avoid unnecessary work when computing PageRanks.

Optimizing parallel PageRank

Question: Can we improve our parallel PageRank code?
Consider: Which PageRanks change significantly (by more than the threshold) after the first iteration?

Iteration 4 significant updates:



Idea: If a page's PageRank converges, don't reprocess it immediately.

- Avoid unnecessary work when computing PageRanks.
- Only process a page if the PageRank of a neighboring page changes.

Optimizing parallel PageRank

Idea: Only process a page if the PageRank of a neighboring page changes.
Problem: How do we efficiently track which pages need to be processed on the next iteration?

Optimizing parallel PageRank

Idea: Only process a page if the PageRank of a neighboring page changes.
Problem: How do we efficiently track which pages need to be processed on the next iteration?
Solution: Use a ***bag***.

Storing a set

A ***bag*** is a multi-set data structure that supports the following special operations:

`Bag_Create()`	Create a new, empty bag.
`Bag_Insert()`	Add an element to a bag.
`Bag_Split()`	Divide a bag into two approximately-equal-size bags.
`Bag_Union()`	Combine the contents of two bags into a single bag.

Idea: Use bags to store vertices to process in each iteration.

Using a bag

Idea: Use bags to store vertices to process in each iteration.

`Bag_Split()` allows for efficient parallel traversal of the elements of the bag.

```
void processBag(Bag<int> *b) {
  if (b->size < threshold) {
    // Process bag's contents serially
  } else {
    // Destructively split the bag
    Bag<int> *b2 = b->Bag_Split();
    cilk_spawn processBag(b);
    processBag(b2);
    cilk_sync;
  }
}
```

Using a bag

Idea: Use bags to store vertices to process in each iteration.

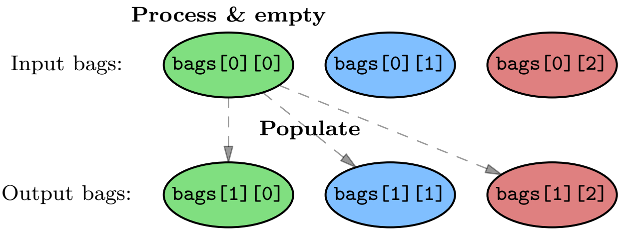
A bag supports parallel insertions when used as a reducer.

```
void processBag(Bag<int> *in,
                Bag_reducer<int> *out) {
  if (b->size < threshold) {
    // Process bag's contents serially
    out->Bag_Insert(/* . . .  */);
  } else {
    // Destructively split the bag
    Bag<int> *in2 = in->Bag_Split();
    cilk_spawn processBag(in, out);
    processBag(in2, out);
    cilk_sync;
  }
}
```

- The bag reducer corresponds to the set monoid $(S, \cup, \emptyset)$, where S is the set of sets.
- `Bag_Union()` implements the reduce operation for the bag reducer.

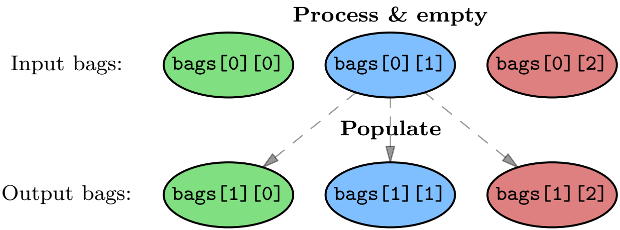
Using bags with coloring

Idea: Use an array of bags such that there are two bags — one “input” and one “output” — for each color.



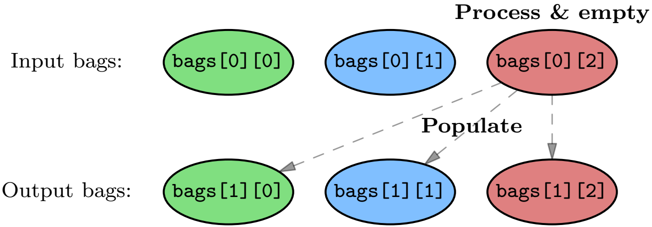
Using bags with coloring

Idea: Use an array of bags such that there are two bags — one "input" and one "output" — for each color.



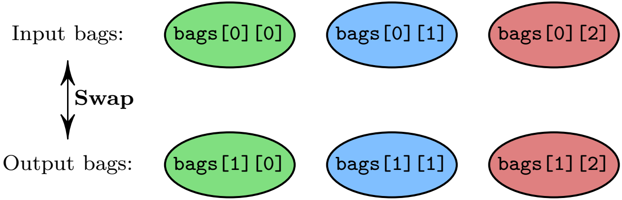
Using bags with coloring

Idea: Use an array of bags such that there are two bags — one “input” and one “output” — for each color.



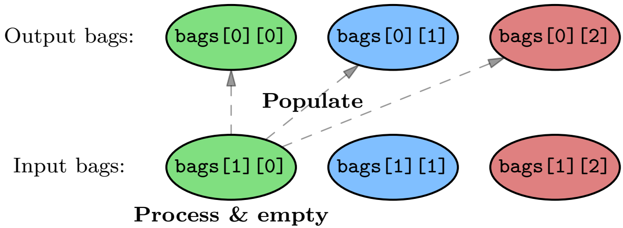
Using bags with coloring

Idea: Use an array of bags such that there are two bags — one "input" and one "output" — for each color.



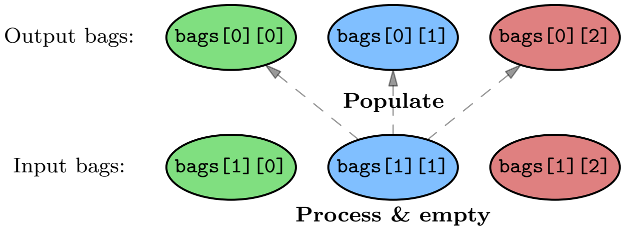
Using bags with coloring

Idea: Use an array of bags such that there are two bags — one “input” and one “output” — for each color.



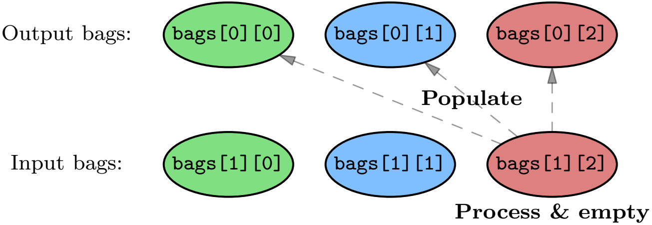
Using bags with coloring

Idea: Use an array of bags such that there are two bags — one “input” and one “output” — for each color.



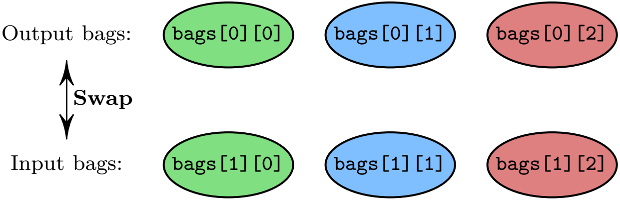
Using bags with coloring

Idea: Use an array of bags such that there are two bags — one “input” and one “output” — for each color.



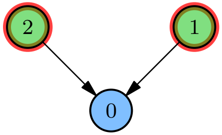
Using bags with coloring

Idea: Use an array of bags such that there are two bags — one "input" and one "output" — for each color.



Using bags with coloring

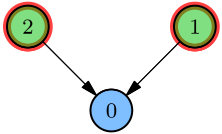
Problem: There is a “race” on inserting a vertex into a bag.



- Both vertices 1 and 2 may attempt to add vertex 0 to their own local view of an output bag.
- This is not technically a determinacy race, but it can cause problems.

Using bags with coloring

Problem: There is a "race" on inserting a vertex into a bag.



- Both vertices 1 and 2 may attempt to add vertex 0 to their own local view of an output bag.
- This is not technically a determinacy race, but it can cause problems.

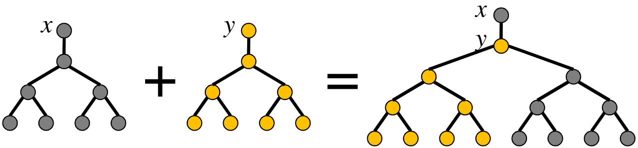
One possible solution: Use a lock or atomic operation to avoid duplicating elements in the bag or processing both duplicates.

- This is nondeterministic code, but
- The input graph is still updated deterministically in a manner consistent with a serial execution.

The bag data structure

Question: How does the bag work?

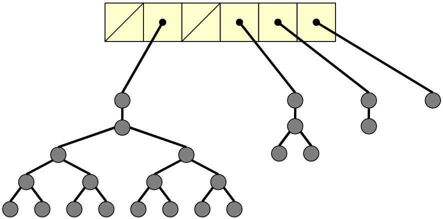
The bag data structure



A bag is made up of ***pennants*** — complete binary trees with extra root nodes — which store the elements.

- Pennants may be split and combined in $\Theta(1)$ time by changing pointers.
- A pennant is only ever combined with another pennant of the same size.

The bag data structure

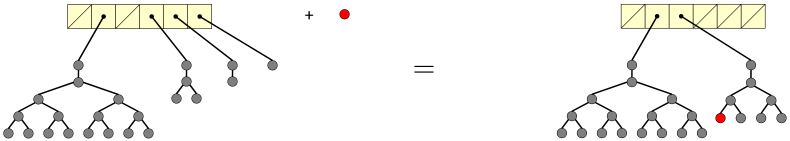


A bag is an array of pointers to pennants.

- The ith entry in the array is either NULL or points to a pennant of size 2^i.
- Intuitively, a bag acts much like a binary counter.

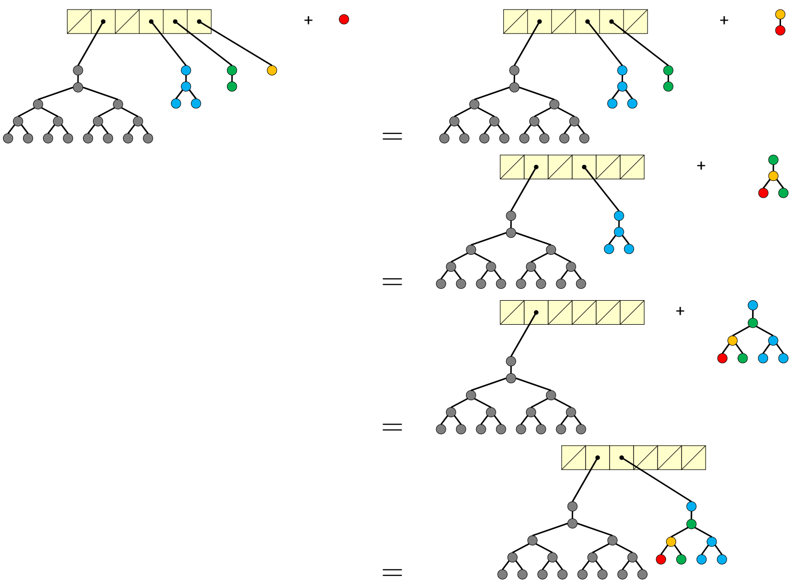
The bag data structure — `Bag_Insert()`

Inserting an element works similarly to incrementing a binary number.



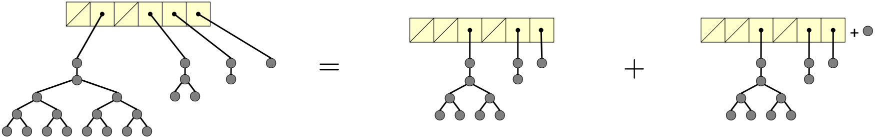
`Bag_Insert()` runs in $O(1)$ amortized time and $O(\lg n)$ worst-case time.

The bag data structure — `Bag_Insert()`



The bag data structure — `Bag_Split()`

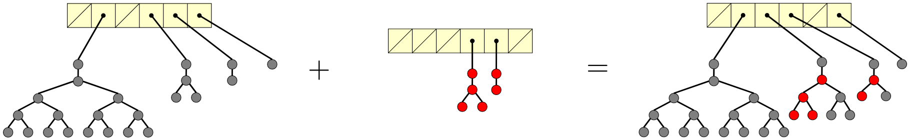
Splitting a bag works similarly to an arithmetic right shift.



`Bag_Split()` runs in $O(\lg n)$ time.

The bag data structure — Bag_Union()

Unioning two bags is works similarly to adding two binary numbers.



Bag_Union() works in $O(\lg n)$ time.

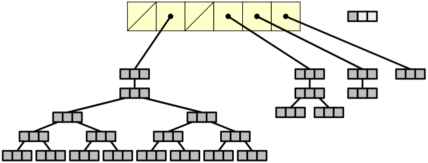
Nondeterminism of bags

Notice: When used as a reducer, the order of elements in a bag is nondeterministic.

- Bags are "logically" deterministic in that the presence of an element in a bag is deterministic.
- Bags encapsulate this nondeterminism and provide the abstraction of an unordered multi-set.

Optimizing the bag data structure

Bags can be made more efficient in practice by storing an array at each node.



- Each node in a pennant stores a fixed-size array of data, which is guaranteed to be full.
- The bag stores an extra fixed-size array of data, called the ***hopper***, which may not be full.
- Inserts first attempt to insert into the hopper. Once the hopper is full, a new, empty hopper is created while the old hopper is inserted into the bag using the original algorithm.

With this optimization, the common case for `Bag_Insert()` is identical to pushing an element onto a FIFO queue.

Performance of parallel PageRank

Actual performance of PageRank on a "power law" graph of 1M vertices and 10M edges (both perform 1.25×10^7 updates):

Version	T_1 *(s)*	T_{12} *(s)*
Serial	28.7	
Chromatic	33.9	4.27

Breakdown of parallel PageRank performance (11 colors used):

	T_1 *(s)*	T_{12} *(s)*
Coloring	3.25	0.67
Iterations	30.60	3.60

Outline

To goal of `fluidanimate` is to solve the problem:

Problem

Simulate the flow of a fluid over time.

To simulate the flow of a fluid, `fluidanimate` uses ***smoothed-particle hydrodynamics***, which

- divides the fluid into discrete units, called ***particles***, and
- approximates any physical property of the system by summing over the pairwise interactions of nearby particles.

fluidanimate

Using smoothed-particle hydrodynamics, `fluidanimate` simulates a fluid flow as follows.

Pseudocode

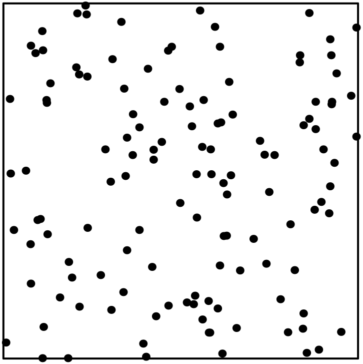
1. For each particle, approximate the physical properties — forces, density, vorticity, velocity, etc. — on that particle.
2. Use these velocities to move each particle over a small time step.
3. Repeat.

Approximately 90% of the total execution time of `fluidanimate` is spent executing inside Step 1. Let's parallelize this step!

fluidanimate

Simplified problem statement

Given a set of particles in space that interact pairwise with nearby particles only, compute all of their pairwise interactions $f(a, b)$.



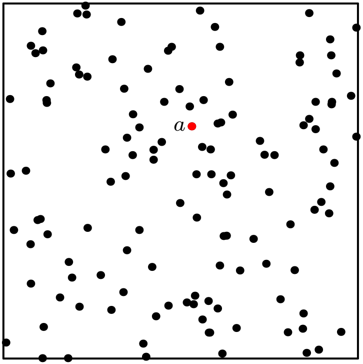
For any two particles a and b, their interaction $f(a, b)$ has the following properties.

- $f(a, b)$ is nonnegligible only if a and b are physically close.
- $f(a, b)$ is symmetric: $f(a, b) = -f(b, a)$.
- $f(a, b)$ takes $\Theta(1)$ time to compute in theory.
- $f(a, b)$ is expensive to compute in practice.

fluidanimate

Simplified problem statement

Given a set of particles in space that interact pairwise with nearby particles only, compute all of their pairwise interactions $f(a, b)$.

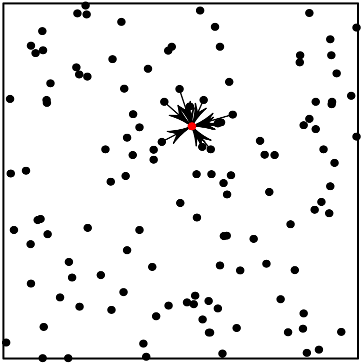


For any two particles a and b, their interaction $f(a, b)$ has the following properties.

- $f(a, b)$ is nonnegligible only if a and b are physically close.
- $f(a, b)$ is symmetric: $f(a, b) = -f(b, a)$.
- $f(a, b)$ takes $\Theta(1)$ time to compute in theory.
- $f(a, b)$ is expensive to compute in practice.

Simplified problem statement

Given a set of particles in space that interact pairwise with nearby particles only, compute all of their pairwise interactions $f(a, b)$.



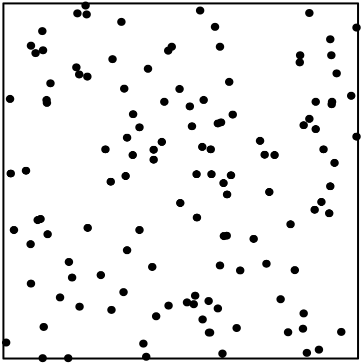
For any two particles a and b, their interaction $f(a, b)$ has the following properties.

- $f(a, b)$ is nonnegligible only if a and b are physically close.
- $f(a, b)$ is symmetric: $f(a, b) = -f(b, a)$.
- $f(a, b)$ takes $\Theta(1)$ time to compute in theory.
- $f(a, b)$ is expensive to compute in practice.

fluidanimate

Simplified problem statement

Given a set of particles in space that interact pairwise with nearby particles only, compute all of their pairwise interactions $f(a, b)$.

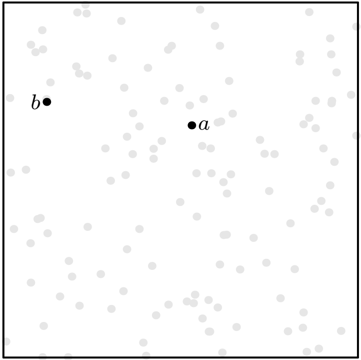


For any two particles a and b, their interaction $f(a, b)$ has the following properties.

- $f(a, b)$ is nonnegligible only if a and b are physically close.
- $f(a, b)$ is symmetric: $f(a, b) = -f(b, a)$.
- $f(a, b)$ takes $\Theta(1)$ time to compute in theory.
- $f(a, b)$ is expensive to compute in practice.

Simplified problem statement

Given a set of particles in space that interact pairwise with nearby particles only, compute all of their pairwise interactions $f(a, b)$.

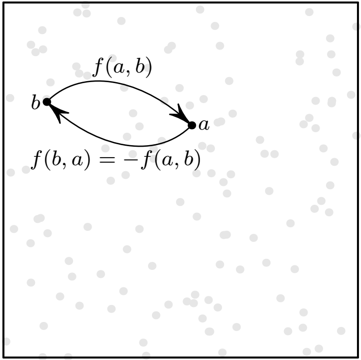


For any two particles a and b, their interaction $f(a, b)$ has the following properties.

- $f(a, b)$ is nonnegligible only if a and b are physically close.
- $f(a, b)$ is symmetric: $f(a, b) = -f(b, a)$.
- $f(a, b)$ takes $\Theta(1)$ time to compute in theory.
- $f(a, b)$ is expensive to compute in practice.

Simplified problem statement

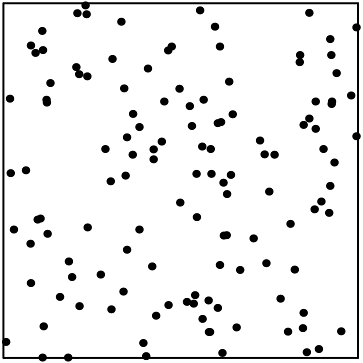
Given a set of particles in space that interact pairwise with nearby particles only, compute all of their pairwise interactions $f(a, b)$.



For any two particles a and b, their interaction $f(a, b)$ has the following properties.

- $f(a, b)$ is nonnegligible only if a and b are physically close.
- $f(a, b)$ is symmetric: $f(a, b) = -f(b, a)$.
- $f(a, b)$ takes $\Theta(1)$ time to compute in theory.
- $f(a, b)$ is expensive to compute in practice.

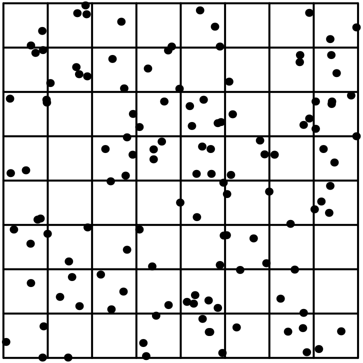
Problem: How do we track which particles are physically close?



Solution: Partition space with a static $\Theta(n) \times \Theta(n)$ grid.

- Expected $\Theta(1)$ particles per grid cell.
- Only consider interactions between particles in same cell and adjacent cells.
- Intuitively, writing to a cell involves reading all cells in the 3×3 square enclosing that cell.

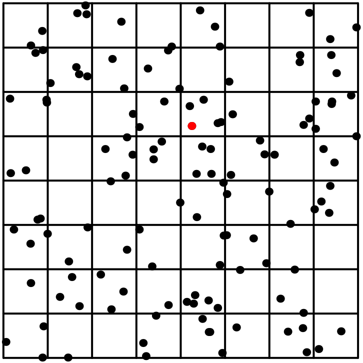
Problem: How do we track which particles are physically close?



Solution: Partition space with a static $\Theta(n) \times \Theta(n)$ grid.

- Expected $\Theta(1)$ particles per grid cell.
- Only consider interactions between particles in same cell and adjacent cells.
- Intuitively, writing to a cell involves reading all cells in the 3×3 square enclosing that cell.

Problem: How do we track which particles are physically close?

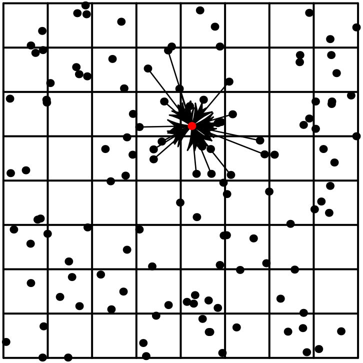


Solution: Partition space with a static $\Theta(n) \times \Theta(n)$ grid.

- Expected $\Theta(1)$ particles per grid cell.
- Only consider interactions between particles in same cell and adjacent cells.
- Intuitively, writing to a cell involves reading all cells in the 3×3 square enclosing that cell.

fluidanimate

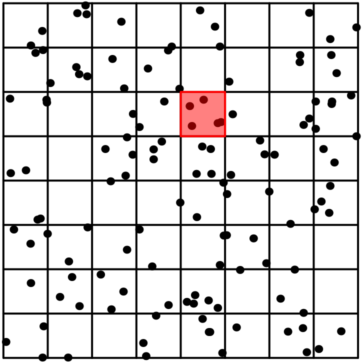
Problem: How do we track which particles are physically close?



Solution: Partition space with a static $\Theta(n) \times \Theta(n)$ grid.

- Expected $\Theta(1)$ particles per grid cell.
- Only consider interactions between particles in same cell and adjacent cells.
- Intuitively, writing to a cell involves reading all cells in the 3×3 square enclosing that cell.

Problem: How do we track which particles are physically close?

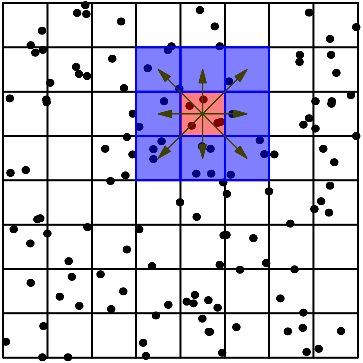


Solution: Partition space with a static $\Theta(n) \times \Theta(n)$ grid.

- Expected $\Theta(1)$ particles per grid cell.
- Only consider interactions between particles in same cell and adjacent cells.
- Intuitively, writing to a cell involves reading all cells in the 3×3 square enclosing that cell.

fluidanimate

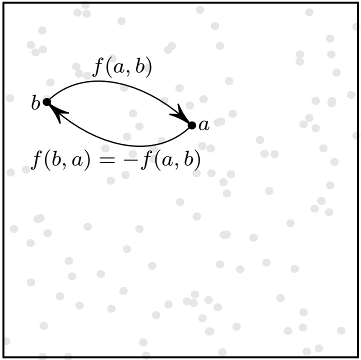
Problem: How do we track which particles are physically close?



Solution: Partition space with a static $\Theta(n) \times \Theta(n)$ grid.

- Expected $\Theta(1)$ particles per grid cell.
- Only consider interactions between particles in same cell and adjacent cells.
- Intuitively, writing to a cell involves reading all cells in the 3×3 square enclosing that cell.

Key optimization: For each pair of particles a and b, compute $f(a, b)$ once, then write $f(a, b)$ to a and $-f(a, b)$ to b.

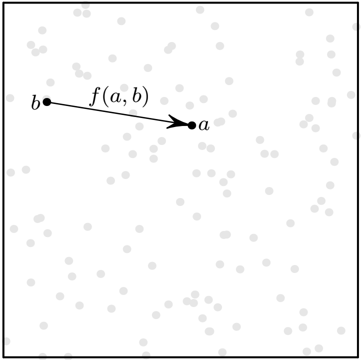


This optimization intuitively cuts the work of computing all interactions $f(a, b)$ in half.

Performance Data:

Optimization	T_1 *(s)*
Without	6.41
With	4.85

Key optimization: For each pair of particles a and b, compute $f(a,b)$ once, then write $f(a,b)$ to a and $-f(a,b)$ to b.

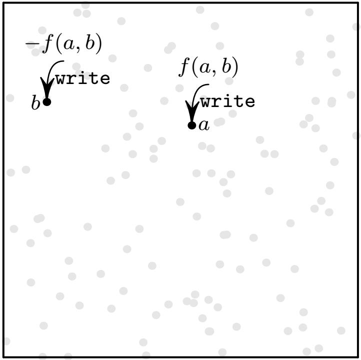


This optimization intuitively cuts the work of computing all interactions $f(a,b)$ in half.

Performance Data:

Optimization	T_1 *(s)*
Without	6.41
With	4.85

Key optimization: For each pair of particles a and b, compute $f(a,b)$ once, then write $f(a,b)$ to a and $-f(a,b)$ to b.



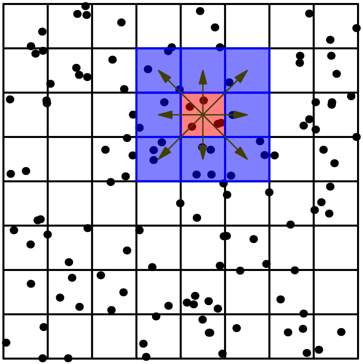
This optimization intuitively cuts the work of computing all interactions $f(a,b)$ in half.

Performance Data:

Optimization	T_1 *(s)*
Without	6.41
With	4.85

fluidanimate

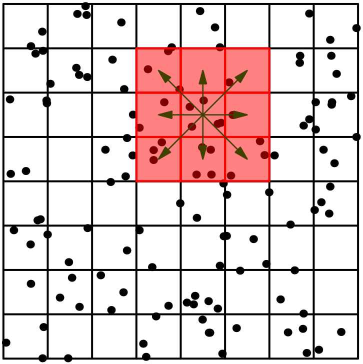
Issue: Updating a cell involves writing to that cell and all neighboring cells.



- Intuitively, all cells in the 3×3 square enclosing a target cell are written.
- Updating a cell updates a *tile* of the grid.

fluidanimate

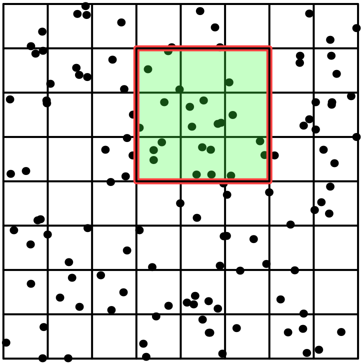
Issue: Updating a cell involves writing to that cell and all neighboring cells.



- Intuitively, all cells in the 3×3 square enclosing a target cell are written.
- Updating a cell updates a *tile* of the grid.

fluidanimate

Issue: Updating a cell involves writing to that cell and all neighboring cells.

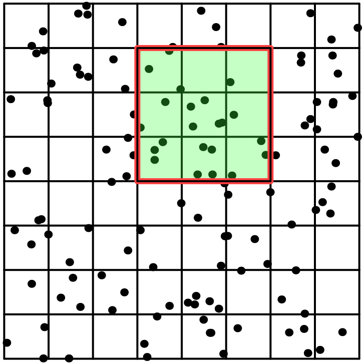


- Intuitively, all cells in the 3×3 square enclosing a target cell are written.
- Updating a cell updates a *tile* of the grid.

Parallel `fluidanimate`

Question

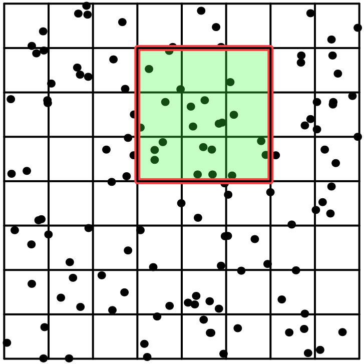
How do we compute these interactions in parallel?



Parallel `fluidanimate`

Question

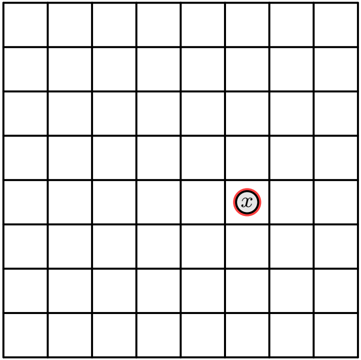
How do we compute these interactions in parallel?



Let's try coloring!

Graph coloring for `fluidanimate`

Idea: Consider the conflict graph.



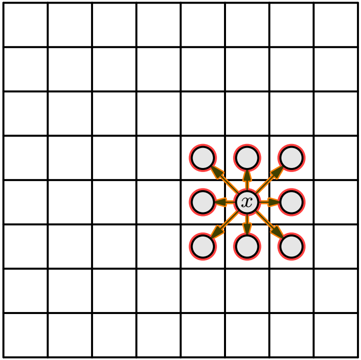
A grid cell conflicts with its neighbors and any cell that writes to one of its neighbors.

- Cell (i, j) may write to any cell (i', j') where $|i - i'| \leq 1$ and $|j - j'| \leq 1$.
- Therefore cell (i, j) conflicts with all cells (i', j') where $|i - i'| \leq 2$ and $|j - j'| \leq 2$.
- Conflict graph has degree 24, and may be colored with 25 colors.

Can we do better?

Graph coloring for `fluidanimate`

Idea: Consider the conflict graph.



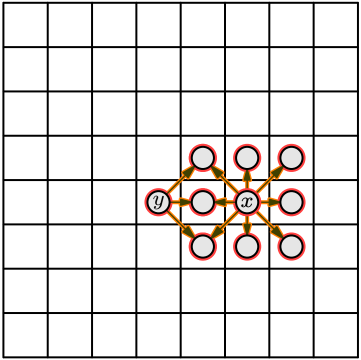
A grid cell conflicts with its neighbors and any cell that writes to one of its neighbors.

- Cell (i, j) may write to any cell (i', j') where $|i - i'| \leq 1$ and $|j - j'| \leq 1$.
- Therefore cell (i, j) conflicts with all cells (i', j') where $|i - i'| \leq 2$ and $|j - j'| \leq 2$.
- Conflict graph has degree 24, and may be colored with 25 colors.

Can we do better?

Graph coloring for `fluidanimate`

Idea: Consider the conflict graph.



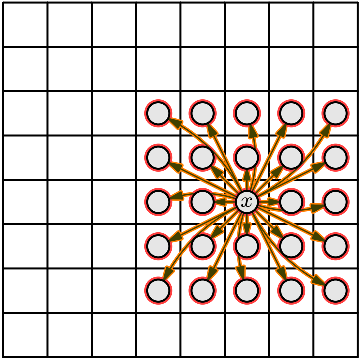
A grid cell conflicts with its neighbors and any cell that writes to one of its neighbors.

- Cell (i, j) may write to any cell (i', j') where $|i - i'| \leq 1$ and $|j - j'| \leq 1$.
- Therefore cell (i, j) conflicts with all cells (i', j') where $|i - i'| \leq 2$ and $|j - j'| \leq 2$.
- Conflict graph has degree 24, and may be colored with 25 colors.

Can we do better?

Graph coloring for `fluidanimate`

Idea: Consider the conflict graph.



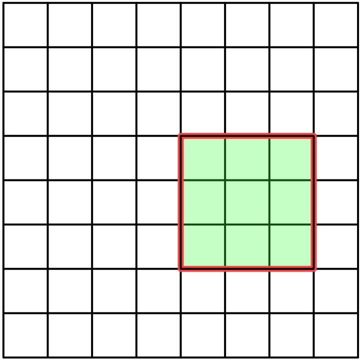
A grid cell conflicts with its neighbors and any cell that writes to one of its neighbors.

- Cell (i, j) may write to any cell (i', j') where $|i - i'| \leq 1$ and $|j - j'| \leq 1$.
- Therefore cell (i, j) conflicts with all cells (i', j') where $|i - i'| \leq 2$ and $|j - j'| \leq 2$.
- Conflict graph has degree 24, and may be colored with 25 colors.

Can we do better?

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?

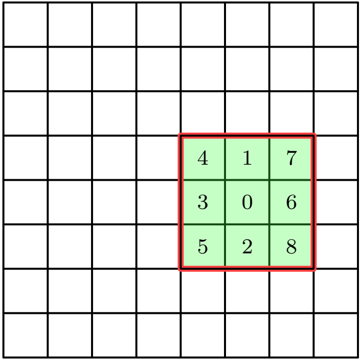


Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?



Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?

7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3

Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?

7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3

Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?

7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3

Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?

7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3

Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?

7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3

Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?

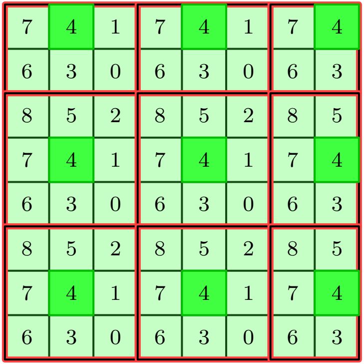
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3

Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?



Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?

7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3

Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?

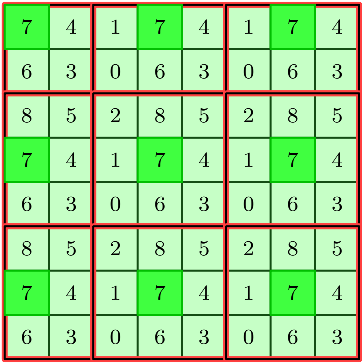
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3

Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?



Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?

7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3

Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Better graph coloring for `fluidanimate`

Question: How many colors do we need for `fluidanimate`?

7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3
8	5	2	8	5	2	8	5
7	4	1	7	4	1	7	4
6	3	0	6	3	0	6	3

Consider first coloring a tile, then tiling the graph.

- Tiling the graph ensures that no tiles conflict.
- Because tiles use a consistent coloring, processing any color processes a tiling of the grid, which induces no conflicts.
- We can color the grid with 9 colors!

Can we do better?

Even better graph coloring for `fluidanimate`

Question: Can we color `fluidanimate` with fewer than 9 colors?

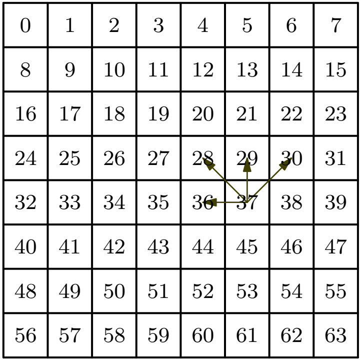
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Suppose each cell only updates itself and cells ***lexicographically smaller*** than itself.

- Each cell updates a set of 5 cells in a new tile.
- Coloring this tile and using it to tile the grid induces a 5-coloring.
- Each color dictates a shifted tiling of the grid, so all nodes of the same color may be processed in parallel.
- We can color the grid with 5 colors!

Even better graph coloring for `fluidanimate`

Question: Can we color `fluidanimate` with fewer than 9 colors?



Suppose each cell only updates itself and cells ***lexicographically smaller*** than itself.

- Each cell updates a set of 5 cells in a new tile.
- Coloring this tile and using it to tile the grid induces a 5-coloring.
- Each color dictates a shifted tiling of the grid, so all nodes of the same color may be processed in parallel.
- We can color the grid with 5 colors!

Even better graph coloring for `fluidanimate`

Question: Can we color `fluidanimate` with fewer than 9 colors?

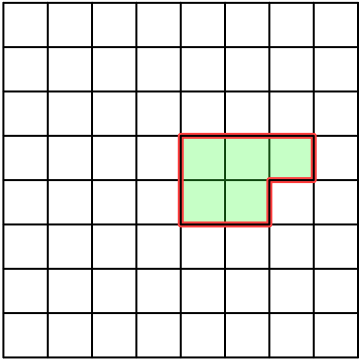
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Suppose each cell only updates itself and cells ***lexicographically smaller*** than itself.

- Each cell updates a set of 5 cells in a new tile.
- Coloring this tile and using it to tile the grid induces a 5-coloring.
- Each color dictates a shifted tiling of the grid, so all nodes of the same color may be processed in parallel.
- We can color the grid with 5 colors!

Even better graph coloring for `fluidanimate`

Question: Can we color `fluidanimate` with fewer than 9 colors?

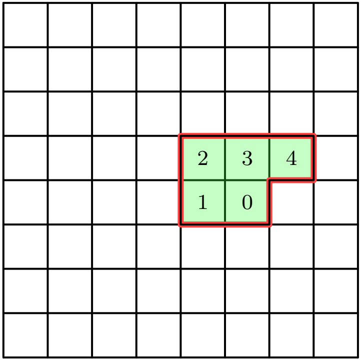


Suppose each cell only updates itself and cells ***lexicographically smaller*** than itself.

- Each cell updates a set of 5 cells in a new tile.
- Coloring this tile and using it to tile the grid induces a 5-coloring.
- Each color dictates a shifted tiling of the grid, so all nodes of the same color may be processed in parallel.
- We can color the grid with 5 colors!

Even better graph coloring for `fluidanimate`

Question: Can we color `fluidanimate` with fewer than 9 colors?

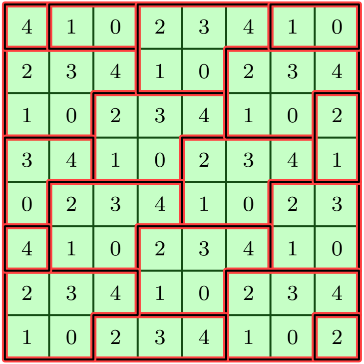


Suppose each cell only updates itself and cells ***lexicographically smaller*** than itself.

- Each cell updates a set of 5 cells in a new tile.
- Coloring this tile and using it to tile the grid induces a 5-coloring.
- Each color dictates a shifted tiling of the grid, so all nodes of the same color may be processed in parallel.
- We can color the grid with 5 colors!

Even better graph coloring for `fluidanimate`

Question: Can we color `fluidanimate` with fewer than 9 colors?



Suppose each cell only updates itself and cells ***lexicographically smaller*** than itself.

- Each cell updates a set of 5 cells in a new tile.
- Coloring this tile and using it to tile the grid induces a 5-coloring.
- Each color dictates a shifted tiling of the grid, so all nodes of the same color may be processed in parallel.
- We can color the grid with 5 colors!

Even better graph coloring for `fluidanimate`

Question: Can we color `fluidanimate` with fewer than 9 colors?

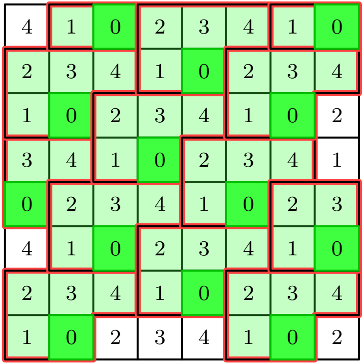
4	1	0	2	3	4	1	0
2	3	4	1	0	2	3	4
1	0	2	3	4	1	0	2
3	4	1	0	2	3	4	1
0	2	3	4	1	0	2	3
4	1	0	2	3	4	1	0
2	3	4	1	0	2	3	4
1	0	2	3	4	1	0	2

Suppose each cell only updates itself and cells ***lexicographically smaller*** than itself.

- Each cell updates a set of 5 cells in a new tile.
- Coloring this tile and using it to tile the grid induces a 5-coloring.
- Each color dictates a shifted tiling of the grid, so all nodes of the same color may be processed in parallel.
- We can color the grid with 5 colors!

Even better graph coloring for `fluidanimate`

Question: Can we color `fluidanimate` with fewer than 9 colors?

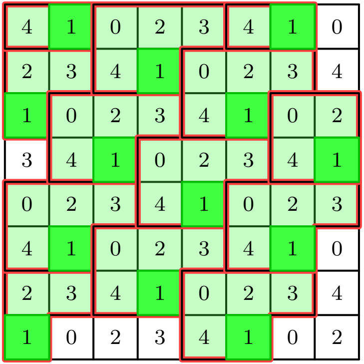


Suppose each cell only updates itself and cells ***lexicographically smaller*** than itself.

- Each cell updates a set of 5 cells in a new tile.
- Coloring this tile and using it to tile the grid induces a 5-coloring.
- Each color dictates a shifted tiling of the grid, so all nodes of the same color may be processed in parallel.
- We can color the grid with 5 colors!

Even better graph coloring for `fluidanimate`

Question: Can we color `fluidanimate` with fewer than 9 colors?

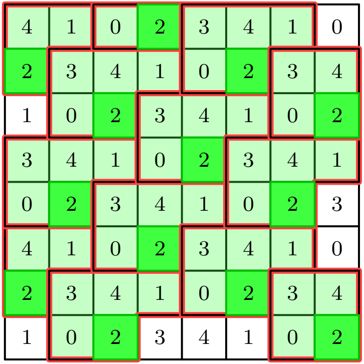


Suppose each cell only updates itself and cells ***lexicographically smaller*** than itself.

- Each cell updates a set of 5 cells in a new tile.
- Coloring this tile and using it to tile the grid induces a 5-coloring.
- Each color dictates a shifted tiling of the grid, so all nodes of the same color may be processed in parallel.
- We can color the grid with 5 colors!

Even better graph coloring for `fluidanimate`

Question: Can we color `fluidanimate` with fewer than 9 colors?

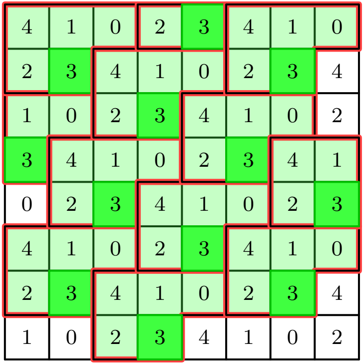


Suppose each cell only updates itself and cells ***lexicographically smaller*** than itself.

- Each cell updates a set of 5 cells in a new tile.
- Coloring this tile and using it to tile the grid induces a 5-coloring.
- Each color dictates a shifted tiling of the grid, so all nodes of the same color may be processed in parallel.
- We can color the grid with 5 colors!

Even better graph coloring for `fluidanimate`

Question: Can we color `fluidanimate` with fewer than 9 colors?

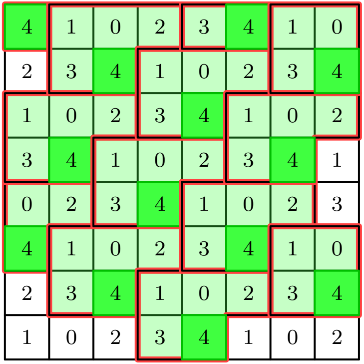


Suppose each cell only updates itself and cells ***lexicographically smaller*** than itself.

- Each cell updates a set of 5 cells in a new tile.
- Coloring this tile and using it to tile the grid induces a 5-coloring.
- Each color dictates a shifted tiling of the grid, so all nodes of the same color may be processed in parallel.
- We can color the grid with 5 colors!

Even better graph coloring for `fluidanimate`

Question: Can we color `fluidanimate` with fewer than 9 colors?



Suppose each cell only updates itself and cells ***lexicographically smaller*** than itself.

- Each cell updates a set of 5 cells in a new tile.
- Coloring this tile and using it to tile the grid induces a 5-coloring.
- Each color dictates a shifted tiling of the grid, so all nodes of the same color may be processed in parallel.
- We can color the grid with 5 colors!

Performance of `fluidanimate`

Theoretical Performance for 2-D `fluidanimate`

Work:

Span:

Performance of `fluidanimate`

Theoretical Performance for 2-D `fluidanimate`
Work: $W(n) = \Theta(n^2)$
Span:

Performance of `fluidanimate`

Theoretical Performance for 2-D `fluidanimate`
Work: $W(n) = \Theta(n^2)$
Span: $S(n) = \langle\text{number of colors}\rangle \cdot (O(\lg n) + \Theta(1)) = O(\lg n)$

Performance of `fluidanimate`

Theoretical Performance for 2-D `fluidanimate`
Work: $W(n) = \Theta(n^2)$
Span: $S(n) = \langle\text{number of colors}\rangle \cdot (O(\lg n) + \Theta(1)) = O(\lg n)$

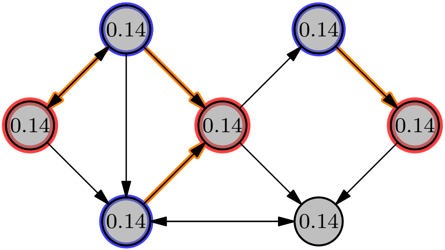
Actual performance for 3-D `fluidanimate` on $300,000$ particles in $135,000$ cells.

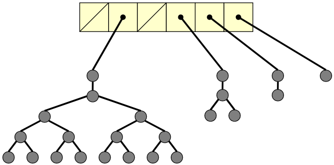
Version	T_1 *(s)*	T_8 *(s)*	*Parallelism*
Cilk, 14-coloring	4.28	0.60	1894
Pthreads	5.32	0.81	
Cilk, "stencil"	6.45	0.82	23791

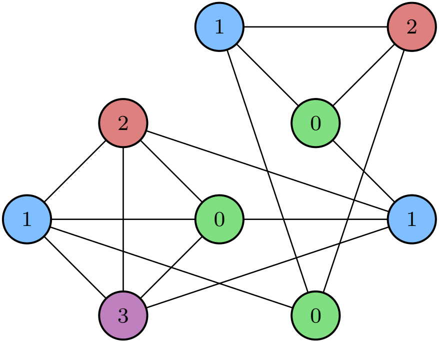
Conclusion

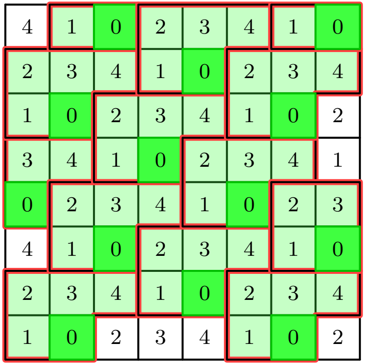
- Chromatic scheduling allows for parallel updates on graphs that produce deterministic results that are consistent with a serial execution.
- Computing a chromatic schedule can be relatively cheap.
- Chromatic schedules can be very efficient.
- Chromatic scheduling can coordinate updating all nodes in the graph in parallel.
- Using bags, chromatic scheduling can support updating nodes in a graph sparsely.

Questions?


0.14
0.14
0.14
0.14
0.14
0.14
0.14
0.14




1
2
0
2
1
0
1
3
0


4 1 0 2 3 4 1 0
2 3 4 1 0 2 3 4
1 0 2 3 4 1 0 2
3 4 1 0 2 3 4 1
0 2 3 4 1 0 2 3
4 1 0 2 3 4 1 0
2 3 4 1 0 2 3 4
1 0 2 3 4 1 0 2