# Раскраска графов для параллельных вычислений и распараллеливание раскраски

Кузьмина Елизавета
371 группа

2017

# Мотивация

- Раскраска графов:
  - вершины — выполняемые операции
  - ребра соединяют вершины, которые имеют общие данные
  - необходимо выявить наборы операций, которые могут вычисляться параллельно
- Распараллеливание раскраски:
  - необходимо ускорить стадию подготовки графа к вычислениям

# Раскраска графов

- Задача нахождения оптимальной раскраски — NP-сложная
- Чем меньше цветов получим — тем оптимальнее можно будет распараллеливать вычисления
- Самый распространенный алгоритм — жадный алгоритм
  - Welsh and Powell
  - Полиномиальное время работы
  - Использует максимум (d + 1) цвет, где d — степень графа
- Эвристические подходы для определения порядка раскраски

# Распараллеливание раскраски

- На основе жадного алгоритма
- На основе максимального независимого множества (Lubu)
  - Jones and Plassmann (JP) алгоритм

**Algorithm 3** Parallel JP Algorithm [7]

```
1:  procedure JP(G(V, E))
2:      W ← V, c ← 1
3:      while W ≠ ∅ do
4:          S ← ∅
5:          for each vertex v ∈ W in parallel do
6:              r(v) ← random()
7:          end for
8:          for each vertex v ∈ W in parallel do
9:              flag ← true
10:             for each vertex w ∈ adj(v) do
11:                 if r(v) <= r(w) then
12:                     flag ← false
13:                 end if
14:             end for
15:             if flag = true then
16:                 S ← S ∪ {v}
17:             end if
18:         end for
19:         for each vertex v ∈ S in parallel do
20:             color[v] ← c
21:         end for
22:         W ← W − S, c ← c + 1
23:     end while
24: end procedure
```

# Serial greedy algorithm

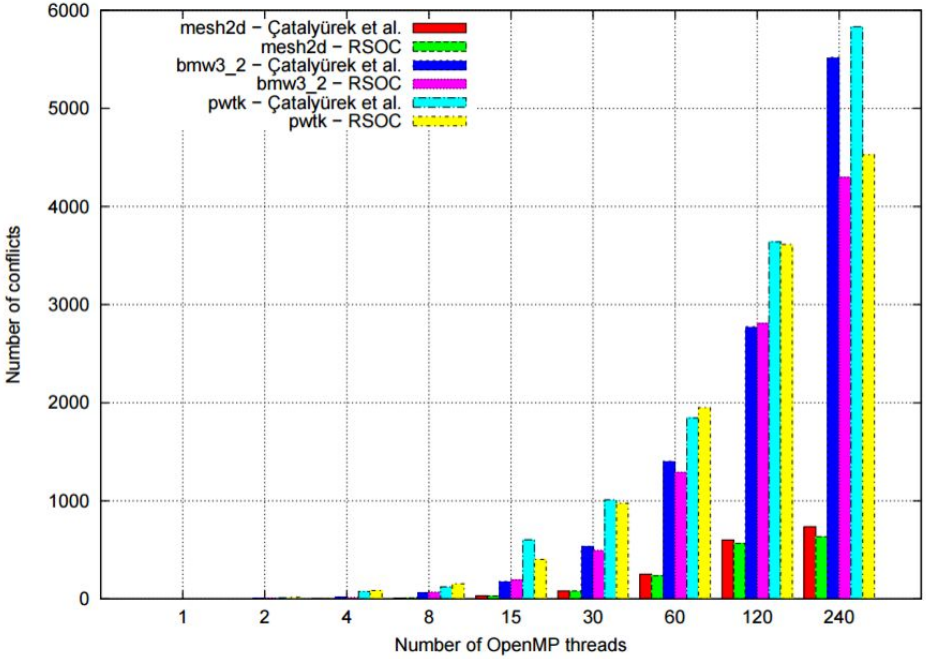
- Gebremedhin and Manne (GM)
- Быстрый и масштабируемый
- Стадии
  - Все вершины раскрашиваются параллельно жадным алгоритмом
  - Параллельно обнаруживаются конфликты
  - Все конфликты решаются в одном потоке
- Catalyurek ускорил алгоритм
  - На втором этапе формируется множество вершин, которые нужно перекрасить, затем алгоритм повторяется

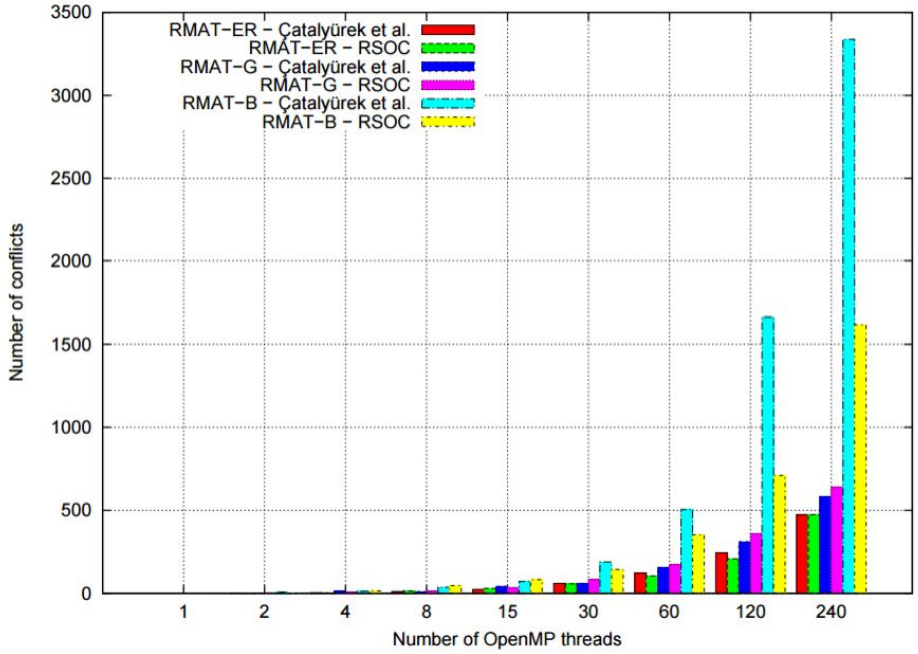
# Rocos algorithm

**Algorithm 3** The improved parallel graph coloring technique.

Input: $\mathcal{G}(V, E)$
**#pragma omp parallel for** ▷ perform tentative coloring on $\mathcal{G}$; round 0
**for all** vertices $V_i \in \mathcal{G}$ **do**
  $\mathcal{C} \leftarrow \{\text{colors of all colored vertices } V_j \in adj(V_i)\}$
  $c(V_i) \leftarrow \{\text{smallest color} \notin \mathcal{C}\}$
**#pragma omp barrier**
$\mathcal{U}^0 \leftarrow V$ ▷ mark all vertices for inspection
$i \leftarrow 1$ ▷ round counter
**while** $\mathcal{U}^{i-1} \neq \emptyset$ **do** ▷ $\exists$ vertices (re-)colored in the last round
  $\mathcal{L} \leftarrow \emptyset$ ▷ global list of defectively colored vertices
  **#pragma omp parallel for**
  **for all** vertices $V_i \in \mathcal{U}^{i-1}$ **do**
    **if** $\exists V_j \in adj(V_i), V_j > V_i : c(V_j) == c(V_i)$ **then** ▷ if they are (still) defective
      $\mathcal{C} \leftarrow \{\text{colors of all colored } V_j \in adj(V_i)\}$ ▷ re-color them
      $c(V_i) \leftarrow \{\text{smallest color} \notin \mathcal{C}\}$
      $\mathcal{L} \leftarrow \mathcal{L} \cup V_i$ ▷ $V_i$ was re-colored in this round
  **#pragma omp barrier**
  $\mathcal{U}_i \leftarrow \mathcal{L}$ ▷ Vertices to be inspected in the next round
  $i \leftarrow i + 1$ ▷ proceed to the next round

| | | Intel® Xeon® | | | | | | Intel® Xeon Phi™ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Number of OpenMP threads | | | | | | Number of OpenMP threads | | | | | | | | |
| | | 1 | 2 | 4 | 8 | 16 | 32 | 1 | 2 | 4 | 8 | 15 | 30 | 60 | 120 | 240 |
| mesh2d | C: | 62.7 | 34.0 | 19.2 | 10.2 | 5.92 | 4.28 | 496 | 252 | 127 | 64.9 | 35.5 | 19.0 | 11.7 | 12.7 | 73.6 |
| | R: | 62.2 | 31.3 | 17.7 | 9.42 | 5.50 | 4.05 | 495 | 249 | 125 | 63.3 | 34.5 | 17.9 | 10.7 | 10.5 | 69.4 |
| bmw3_2 | C: | 58.1 | 33.5 | 14.4 | 7.84 | 4.73 | 3.61 | 468 | 235 | 118 | 60.0 | 33.1 | 18.0 | 11.5 | 12.7 | 74.2 |
| | R: | 57.8 | 29.4 | 12.1 | 6.48 | 3.91 | 3.30 | 466 | 234 | 117 | 59.2 | 32.4 | 17.1 | 9.88 | 11.0 | 54.9 |
| pwtk | C: | 40.1 | 24.0 | 14.5 | 8.07 | 4.96 | 3.65 | 465 | 233 | 117 | 59.6 | 33.2 | 18.2 | 11.1 | 12.9 | 74.4 |
| | R: | 39.8 | 20.0 | 11.3 | 6.08 | 3.81 | 3.30 | 464 | 232 | 117 | 58.9 | 32.4 | 17.2 | 10.6 | 11.0 | 59.9 |
| RMAT-ER | C: | 6.11 | 3.21 | 1.82 | 1.09 | 0.79 | 0.85 | 196 | 97.8 | 48.9 | 24.6 | 13.0 | 6.41 | 3.16 | 1.64 | 0.94 |
| | R: | 6.09 | 3.20 | 1.81 | 1.08 | 0.78 | 0.85 | 196 | 98.0 | 49.0 | 24.7 | 13.1 | 6.43 | 3.16 | 1.64 | 0.95 |
| RMAT-G | C: | 6.10 | 3.18 | 1.82 | 1.08 | 0.77 | 0.81 | 195 | 97.1 | 48.6 | 24.3 | 12.9 | 6.34 | 3.12 | 1.62 | 0.93 |
| | R: | 6.07 | 3.17 | 1.81 | 1.07 | 0.77 | 0.81 | 195 | 97.3 | 48.7 | 24.4 | 13.0 | 6.38 | 3.13 | 1.63 | 0.93 |
| RMAT-B | C: | 5.47 | 2.86 | 1.62 | 0.93 | 0.65 | 0.64 | 189 | 94.1 | 46.7 | 23.5 | 12.3 | 6.08 | 3.12 | 1.90 | 1.49 |
| | R: | 5.46 | 2.83 | 1.60 | 0.92 | 0.64 | 0.63 | 189 | 94.0 | 46.9 | 23.5 | 12.4 | 6.02 | 2.95 | 1.60 | 1.00 |


mesh2d − Çatalyürek et al.
mesh2d − RSOC
bmw3_2 − Çatalyürek et al.
bmw3_2 − RSOC
pwtk − Çatalyürek et al.
pwtk − RSOC
Number of conflicts
0
1000
2000
3000
4000
5000
6000
1
2
4
8
15
30
60
120
240
Number of OpenMP threads


RMAT−ER − Çatalyürek et al.
RMAT−ER − RSOC
RMAT−G − Çatalyürek et al.
RMAT−G − RSOC
RMAT−B − Çatalyürek et al.
RMAT−B − RSOC
Number of conflicts
0
500
1000
1500
2000
2500
3000
3500
1
2
4
8
15
30
60
120
240
Number of OpenMP threads

# Распараллеливание раскраски на GPU

- Управление аппаратными ресурсами и сложной иерархией памяти
- Реализации и для жадного алгоритма, и для алгоритма максимального независимого множества
- Оптимизация конкретных алгоритмов так же важна, как и общие методы оптимизации

# Grosset Algorithm

- GM + CUDA
- Три стадии:
  - разбиение на подграфы и идентификация граничных вершин
  - раскраска графа и обнаружение конфликтов
  - последовательное разрешение конфликтов на CPU

# Процедура csrcolor

- Включена в библиотеку CUSPARSE (NVIDIA)
- На основе алгоритма JP
- Несколько хэш-функций вместо генератора случайных чисел

Execution Time Speedup
0
1
2
3
4
5
6
3-step GM
csrcolor
Rmat-er
Rmat-g
Hamrle3
thermal2
atmosmodd
G3_circuit
ASIC_320ks
(a)
Number of Colors
0
20
40
60
80
100
120
3-step GM
csrcolor
Rmat-er
Rmat-g
Hamrle3
thermal2
atmosmodd
G3_circuit
ASIC_320ks
(b)

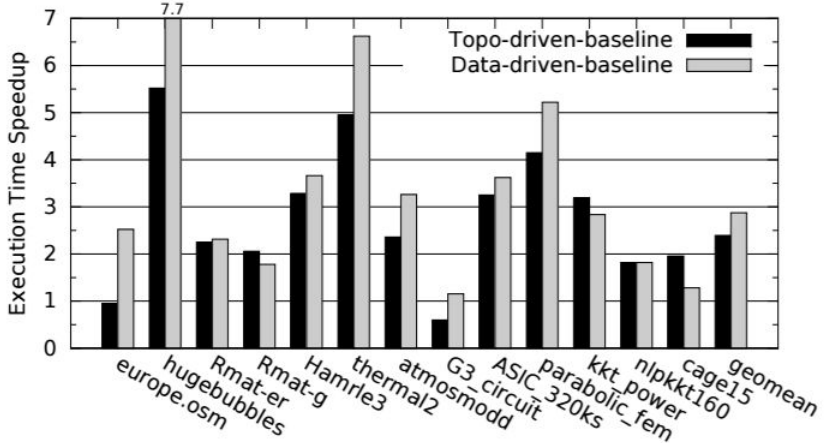
# Chen algorithm

- На основе алгоритма Grosset
- Вся работа производится на GPU, чтобы избежать передачи данных между GPU и CPU

# Chen algorithm

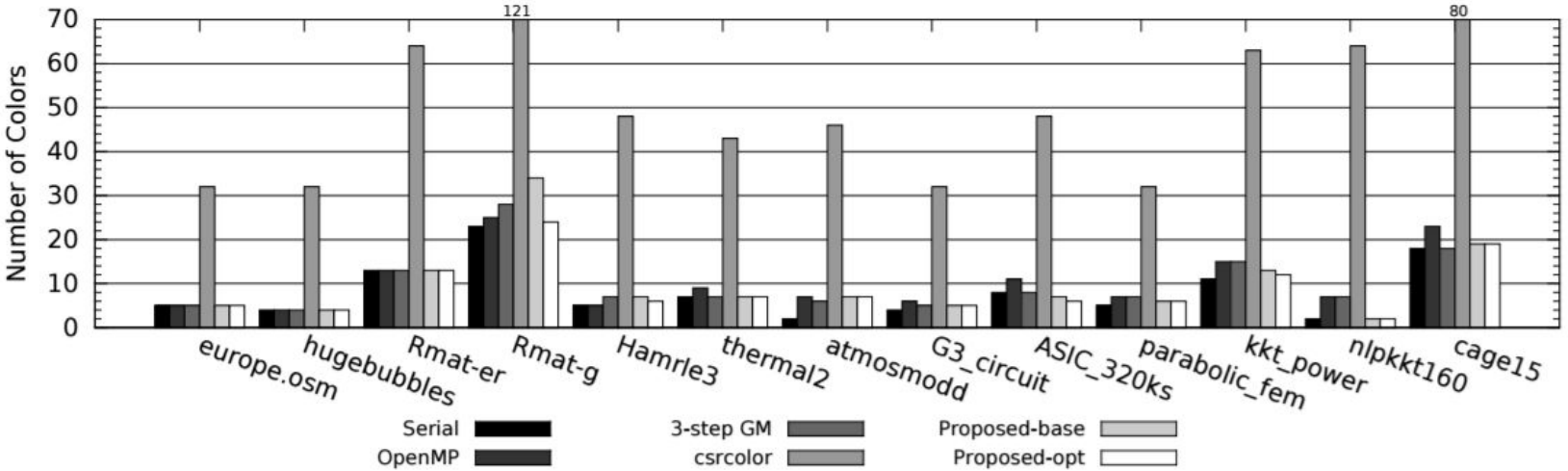
Концепции Nasre

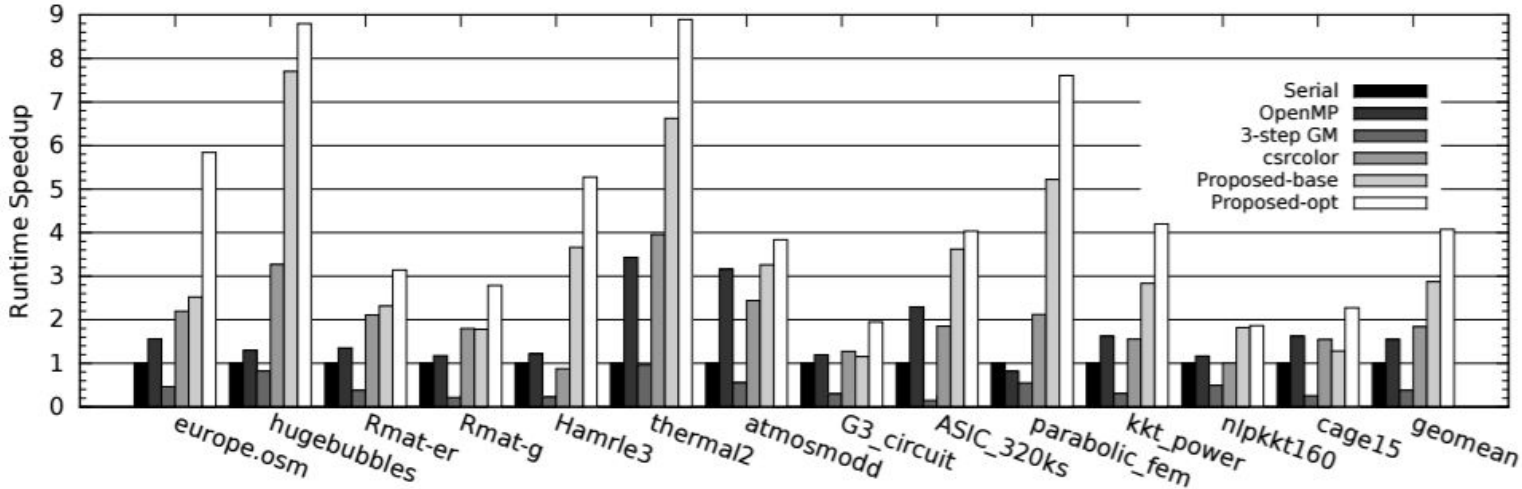
- Концепция управления топологией
  - обработка всех вершин
- Концепция управления данными
  - поддержка списка вершин для обработки
  - использование двух буферов во избежание копирования рабочего списка вершин
  - Merrill — для каждого блока только одна атомная операция

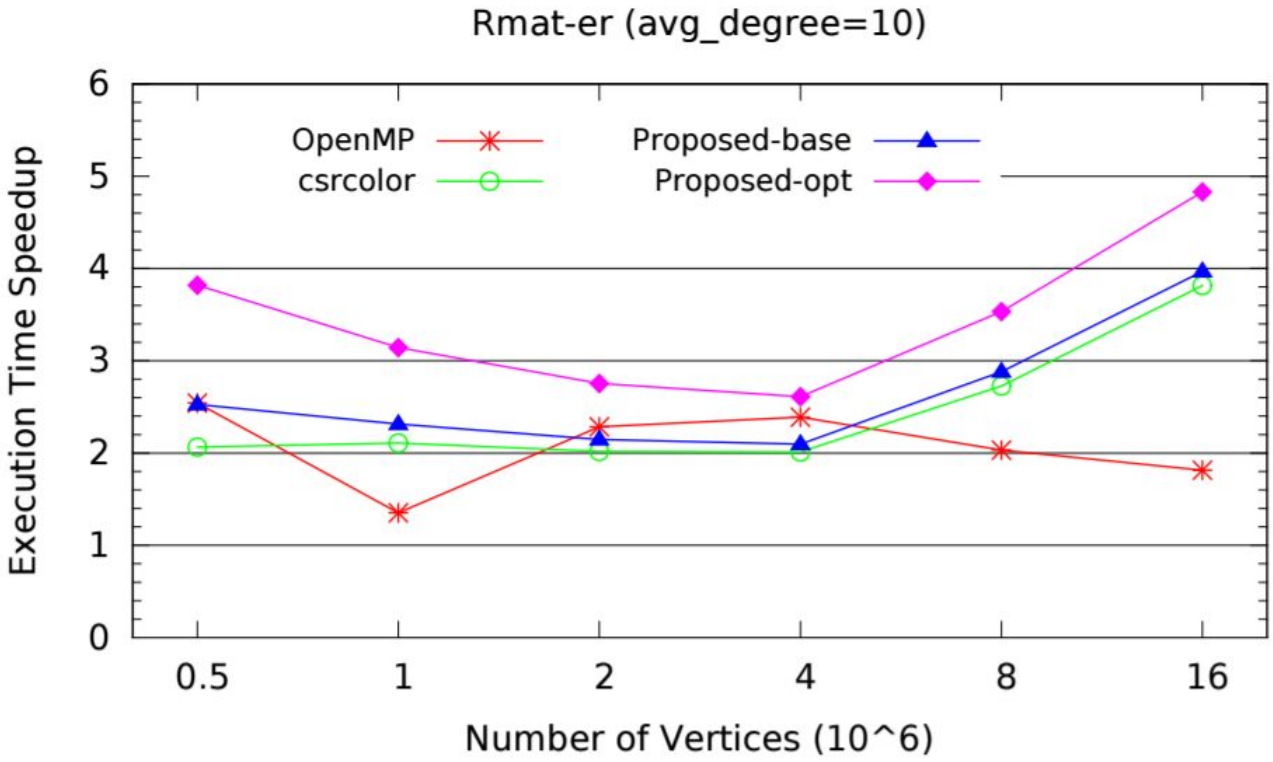


# Chen algorithm

- Борьба с конфликтами
  - Перекрашивание вершины с меньшей степенью — повышение производительности на 10,3%
  - “Утолщение” потоков — повышение на 4,4%
- Применение битовой операции для жадной раскраски — повышение на 61% (повышение производительности всей программы на 28%)
- Ядро Fusion - повышение на 10%
- Кэширование данных только для чтения — повышение на 3,6%
- Балансировка нагрузок (Merrill) — повышение на 6,4%


Number of Colors
70
60
50
40
30
20
10
0
121
80
europe.osm
hugebubbles
Rmat-er
Rmat-g
Hamrle3
thermal2
atmosmodd
G3_circuit
ASIC_320ks
parabolic_fem
kkt_power
nlpkkt160
cage15
Serial
OpenMP
3-step GM
csrcolor
Proposed-base
Proposed-opt


Runtime Speedup
9
8
7
6
5
4
3
2
1
0
europe.osm
hugebubbles
Rmat-er
Rmat-g
Hamrle3
thermal2
atmosmodd
G3_circuit
ASIC_320ks
parabolic_fem
kkt_power
nlpkkt160
cage15
geomean
Serial
OpenMP
3-step GM
csrcolor
Proposed-base
Proposed-opt

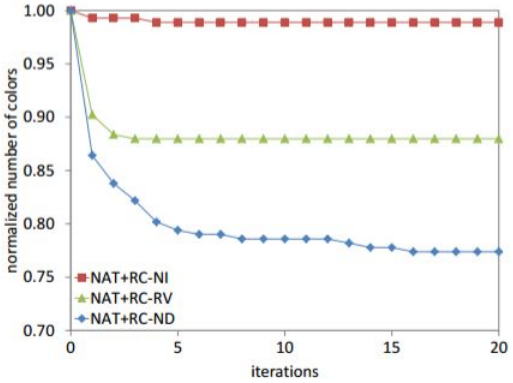

Rmat-er (avg_degree=10)
Execution Time Speedup
Number of Vertices (10^6)
OpenMP
csrcolor
Proposed-base
Proposed-opt
0
1
2
3
4
5
6
0.5
1
2
4
8
16

# Раскраска распределенных графов

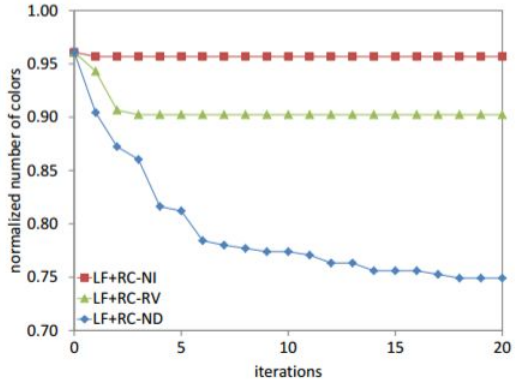
- Граф разбит на подграфы, которые лежат на разных узлах
- Вершины узла делятся на внутренние и граничные
- Bozdag — первая масштабируемая распараллеленная раскраска графов с распределенной памятью
  - жадный алгоритм на каждом узле
  - синхронное разрешение конфликтов

# Sarıyuce algorithm

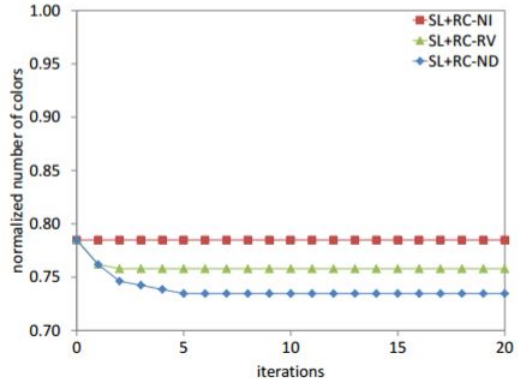
- Эвристика Largest First — O(|V|)
- Эвристика Smallest Last — O(|E|)
- Использование Iterated Greedy (Culberson) для лучшей раскраски
  - RC — перекрашивание вершин
  - aRC — асинхронное перекрашивание
  - Перестановки цветовых классов
    - RV — обратный порядок цветов
    - NI — убывающее число вершин
    - ND — возрастающее число вершин
  - Оптимизация отправки сообщений между процессорами




(a) Natural (NAT)




(b) Largest First (LF)




(c) Smallest Last (SL)