


Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики

ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ГРАФОВ

Лекция 6. Алгоритмы раскраски графа

Пирова А.Ю.
Кафедра МОСТ

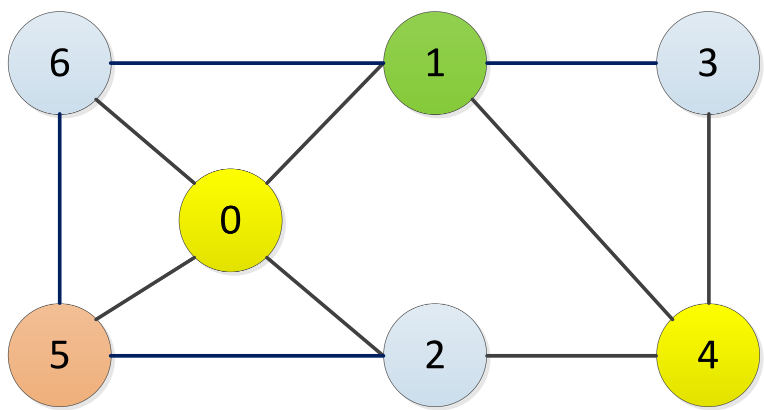
Содержание

Предметные области

- Составление расписаний
- Аллокация регистров
- Вспомогательный этап при распараллеливании алгоритмов разреженной алгебры: вычисление предобуславливателя, итерационное решение СЛАУ, вычисление собственных векторов, определение Якобиана и Гессиана матрицы.
- Вспомогательный этап при распараллеливании алгоритмов на графах: вычисление PageRank, разделение графа и др.

Постановка задачи

- Пусть дан ненаправленный граф $G = (V, E)$, $|V| = n, |E| = m$
- Функция $\sigma: V \rightarrow \{1, \dots, s\}$ называется *раскраской графа*, если для любых смежных вершин $\sigma(v) \neq \sigma(u)$, $(u, v) \in E$
- Минимально возможное число цветов s называется *хроматическим числом графа*
- Задача раскраски графа в минимальное число цветов NP-трудная для не планарных графов



Постановка задачи

- Связанные задачи:
  - 2-раскраска графа: функция $\sigma_2 : V \to \{1, \dots, s_2\}$ называется *раскраской графа*, если для любых вершин u и v, связанных кратчайшим путем не более, чем из двух ребер, $\sigma_2(v) \neq \sigma_2(u)$
  - Раскраска двудольного графа
  - Раскраска динамического графа

Последовательный алгоритм

- Последовательные алгоритмы основаны на жадной стратегии.
- Вершины рассматриваются в некотором порядке, по которому можно построить функцию приоритета $p: V \to \boldsymbol{R}$
- Общая схема:

1. Множество нераскрашенных вершин $U = V$
2. Для $i = 1$ до n:
   1. Выбрать $v_i \in U$ согласно приоритету $p(v_i)$
   2. Выбрать цвет c вершины v_i:
      1. Множество доступных цветов $C = \{1, 2, \dots., \Delta + 1\}$
      2. Для всех $u \in Adj(v_i)$: $p(u) > p(v_i)$ $C = C \setminus \mathrm{c}(u)$
      3. $c(v_i) = \min C$
   3. $U = U \setminus \mathrm{v}_i$

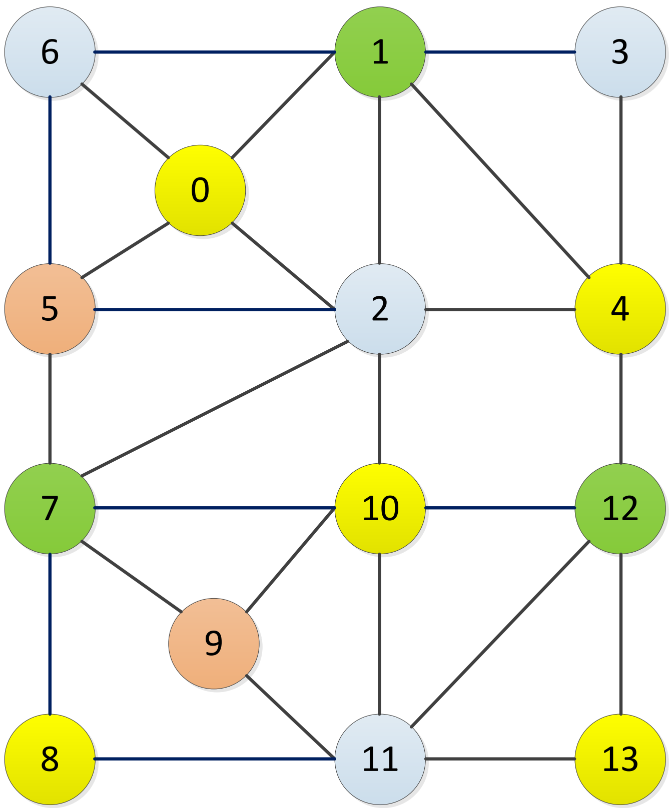
Последовательный алгоритм. Функции приоритета

- First Fit: вершины рассматриваются в некотором заранее выбранном порядке. Например, в порядке появления в графе
- Saturation-Degree-Ordering: выбирается вершина с максимальной насыщенной степенью, то есть с максимальным числом различно раскрашенных смежных вершин.
- Incidence-Degree-Ordering: выбирается вершина с максимальным числом раскрашенных смежных вершин.
- Largest-Degree-First: вершины раскрашиваются в порядке уменьшения степеней
- Smallest-Degree-Last: из графа удаляются все вершины с минимальной степенью, затем рекурсивно раскрашивается оставшийся граф. Удаленные вершины раскрашиваются в последнюю очередь.

Последовательный алгоритм

- Число различных цветов, использованных последовательным алгоритмом, ограничено $\Delta + 1$, где Δ – максимальная степень вершин графа.
- По увеличению качества раскраски: FF, LDF, IDO, SatDO
- Вычислительная сложность для графа с n вершинами и m ребрами: FF - $O(m)$, LDF и IDO - можно реализовать за $O(m)$, SDO - $O(n^2)$.
- Жадный алгоритм плохо распараллеливается

Пример



Порядок раскраски Largest degree first:
2, 1, 7, 10, 11, 0, 4, 5, 12, 6, 9, 3, 8, 13
Номера цветов: 1 - голубой, 2 - зеленый, 3 - желтый, 4 - розовый

Сравнение алгоритмов

	C							T_S						
Graph	FF	R	LF	ID	SL	SD	*Spark*	FF	R	LF	ID	SL	SD	*Spark*
com-orkut	175	132	87	86	83	76		2.23	3.39	3.54	44.13	10.59	46.60	
soc-LiveJournal1	352	330	323	325	322	326		0.89	2.05	2.34	17.93	4.69	19.75	
europe-osm	5	5	4	4	3	3		1.32	13.36	17.15	48.59	19.87	52.73	
cit-Patents	17	21	14	14	13	12		0.50	1.62	2.00	9.82	3.21	10.08	
as-skitter	103	81	71	72	70	70		0.24	1.70	2.43	9.41	2.79	9.94	
wiki-Talk	102	85	72	57	56	51		0.09	0.35	0.49	2.79	0.61	2.90	
web-Google	44	44	45	45	44	44		0.09	0.22	0.25	1.68	0.47	1.77	
com-youtube	57	46	32	28	28	26		0.06	0.19	0.25	1.50	0.35	1.55	
constant1M-50	33	32	32	34	34	26		0.90	1.13	1.16	16.07	2.96	17.23	
constant500K-100	52	52	52	55	53	44		0.74	0.88	0.84	14.20	1.97	15.51	
graph500-5M	220	220	159	157	158	147		1.83	3.14	3.69	25.19	8.43	35.29	
graph500-2M	206	208	153	152	153	141		0.52	0.77	0.98	8.09	2.22	11.68	
rMat-ER-2M	12	12	11	11	11	8		0.47	0.93	1.07	10.10	2.22	9.13	
rMat-G-2M	27	27	15	15	15	11		0.48	0.92	1.18	9.17	2.59	9.07	
rMat-B-2M	105	105	67	67	67	59		0.50	0.83	1.00	8.44	2.41	8.64	
big3dgrid	4	7	7	4	7	5		0.41	3.34	4.07	13.61	4.77	15.30	
clique-chain-400	399	399	399	399	399	399		0.05	0.05	0.05	0.81	0.08	2.06	
path-10M	2	3	3	2	2	2		0.18	1.95	2.49	7.34	2.58	7.96	

Figure 9: Performance measurements for six serial ordering heuristics used by GREEDY, where measurements for real-world graphs appear above the center line and those for synthetic graphs appear below. The columns under the heading C present the average number of colors obtained by each ordering heuristic. The columns under the heading T_S present the average serial running time for each heuristic. The "*Spark*" columns under the C and T_S headings contain bar graphs that pictorially represent the coloring quality and serial running time, respectively, for each of the ordering heuristics. The height of the bar for the coloring quality C_H of ordering heuristic H is proportional to C_H. The bar heights are similar for T_S except that the log of times are used. Section 6 details the experimental setup and graph suite used.

Hasenplaugh W. et al. Ordering heuristics for parallel graph coloring //Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures. – ACM, 2014. – P. 166-177.

Параллельные алгоритмы

- Подход 1: найти независимое множество вершин и раскрасить их параллельно. При раскраске вершины v уже известны цвета некоторых соседей, после назначения цвета вершине он больше не изменится. Вершины обрабатываются в порядке приоритета.
  - Алгоритмы Джонса–Плассмана (1991), Олрайта и др. (1994)
- Подход 2 (speculative coloring): раскраска выполняется итерационно. На каждой итерации процессы параллельно определяют цвета для своих локальных нераскрашенных вершин, затем обмениваются результатами и исправляют ошибки раскраски граничных вершин.
  - Алгоритмы Гебремедхин и Манна (2000), Чаталюрека (2012), Боумана и др. (2005)

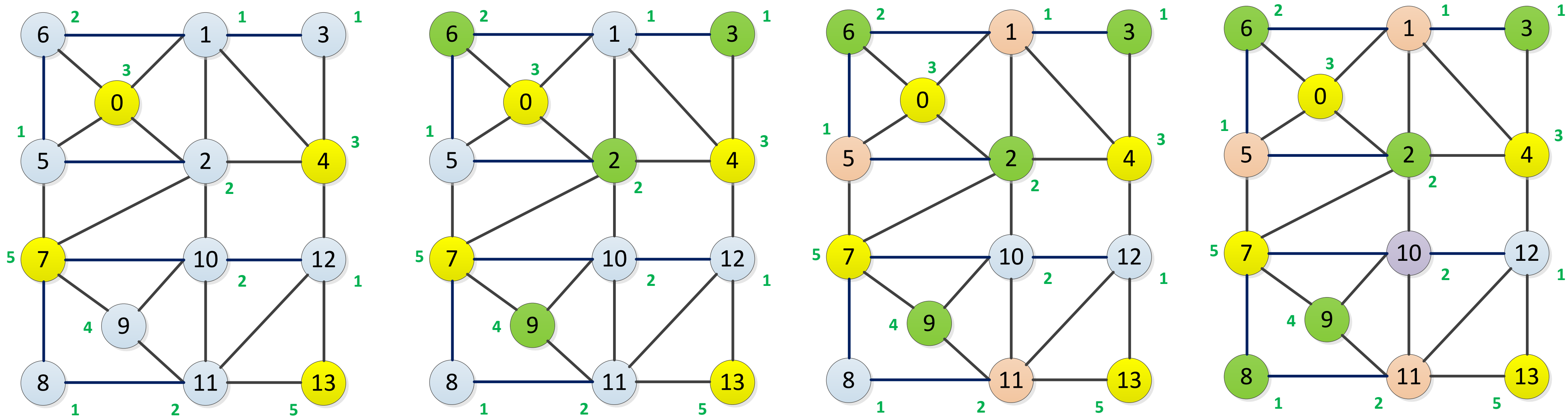
Подход 1. Алгоритм Джонса-Плассмана

- Общая схема:

1. Для каждой вершины $v \in V$ назначить приоритет $w(v)$
2. Множество нераскрашенных вершин $U = V$
3. Пока есть нераскрашенные вершины $U \neq \emptyset$:
   1. Параллельно для всех $v \in U$:
      1. $I = \{z : w(z) > w(u)$ для $\forall u \in Adj(z) \cap U\}$
      2. Параллельно Для всех $z \in I$:
         1. Назначить наименьший доступный цвет $c(z)$
   2. $U = U \backslash I$

- I – независимое множество вершин по правилу Люби: это вершины, чьи случайно сгенерированные веса являются локальным максимумом среди весов их соседей.

Подход 1. Алгоритм Джонса-Плассмана. Пример



Подход 1. Асинхронная реализация

- Асинхронная параллельная реализация:
  – Для каждой вершины определим множества ее соседей с меньшим и с большим приоритетом ($\mathrm{P}rev(v)$ и $\mathrm{N}ext(v)$)
  – Чтобы определить цвет v, необходимо получить цвет всех вершин из $\mathrm{P}rev(v)$. После этого назначается цвет v и отправляется всем вершинам из $\mathrm{N}ext(v)$.
- Приоритет: случайное число (Джонс–Плассман), Largest-Degree-First (Райт), хэш-функция, не требующая коммуникации (Саллинен и др.)…

Подход 1. Асинхронная реализация

1. Для каждой вершины $v \in V$ назначить приоритет $w(v)$
2. Множество нераскрашенных вершин $U = V$
3. Пока есть нераскрашенные вершины $U \neq \emptyset$:
   1. Параллельно для всех $v \in U$:
      1. (Коммуникация) Получить приоритет всех соседей
      2. $\mathrm{P}rev(v) = \{z : w(z) < w(v), (v, z) \in E\}$
      3. $\mathrm{N}ext(v) = \{z : w(z) > w(v), (v, z) \in E\}$
      4. Если $\mathrm{P}rev(v) = \emptyset$, $c(v) = 1$. Иначе:
         1. (Ожидание) Получить цвета всех из $\mathrm{P}rev(v)$, $\mathrm{C}_p(v) = \{c(z) : z \in \mathrm{P}rev(v)\}$
            Доступные цвета: $C = \{1, 2, \dots, |Prev(v)| + 1\}/C_p(v)$
         2. Определить цвет $c(v) = \min(C) + 1$
      5. Отправить цвет $c(v)$ всем из $\mathrm{N}ext(v)$
      6. $U = U/\{v\}$

Вычислительные эксперименты

- Sallinen S. et al. Graph colouring as a challenge problem for dynamic graph processing on distributed systems //SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. – IEEE, 2016. – С. 347-358.

- Инфраструктура: Catalyst cluster, Lawrence Livermore National Laboratory. Процессоры 12-core Intel Xeon E5-2695v2 (2.4 GHz), память 128 GB, Intel 910 PCI-attached NAND Flash на узел.
- Реализация алгоритма раскраски с помощью платформы для систем с распределенной памятью HavoqGT (http://software.llnl.gov/havoqgt/), позволяющей реализовать асинхронные алгоритмы на графах, описанные в терминах операций с вершинами. HavoqGT реализован на языке C++, использует библиотеку Boost.
- Тестовые графы: графы RMAT с числом вершин 2^s , $s = 27,28, \dots 32$ и числом ребер $32 * 2^s$; графы Эрдеша–Реньи, веб-графы.

Вычислительные эксперименты

- Сильная и слабая масштабируемость по фазам:
  - Collect – фаза сбора приоритетов вершин-соседей, Color – фаза назначения цветов вершинам

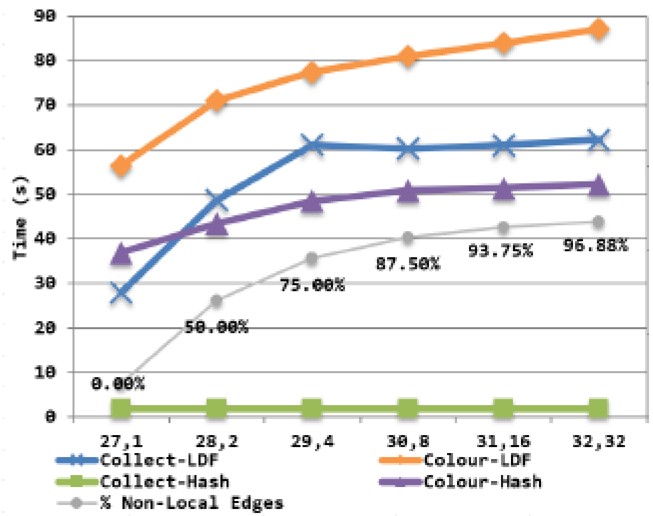


Fig. 1: Weak Scaling Experiments using Erdõs-Rényi graphs. The plot presents runtime for each stage of the algorithm for LDF and Hash priorities. X-axis labels represent $[scale, nodes]$. Also shown is the percentage of edges that are non local for a given node count. A flat line indicates perfect weak scaling.

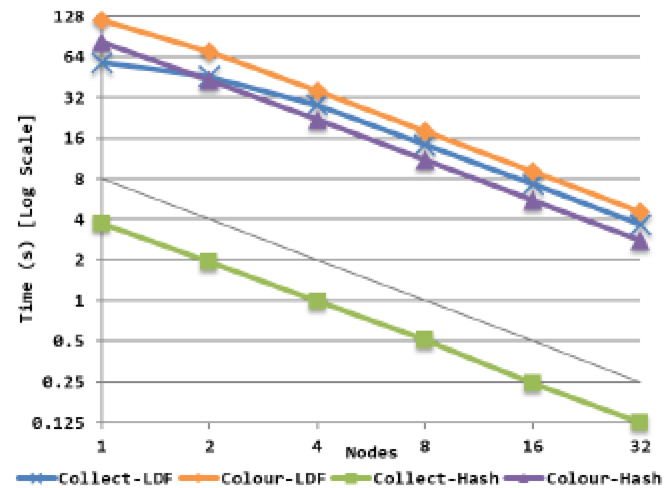


Fig. 2: Strong Scaling Experiment using an Erdõs-Rényi Scale 28 graph. Y-axis represents runtime (log-scale, unlike the weak scaling plot). X-axis represents number of nodes used. A straight diagonal line parallel with the grey line indicate perfect strong scaling.

Вычислительные эксперименты

- Производительность

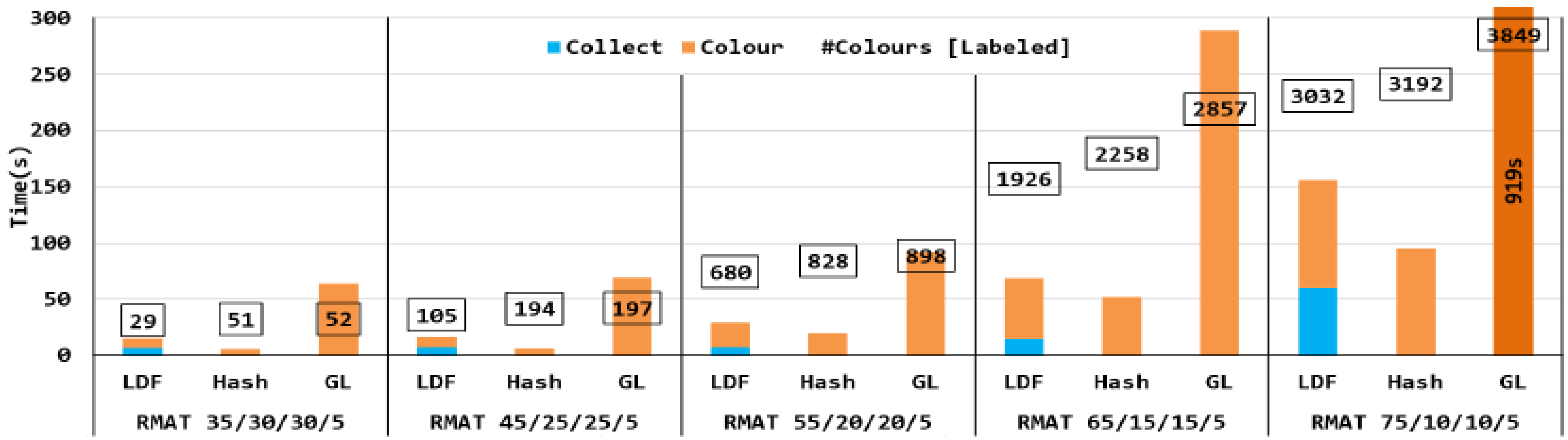


Fig. 3: Run-time and number of colours for RMAT graphs with varying power-law nature (less-pronounced – left, to more-pronounced – right). The experiment uses 8 nodes and a Scale 27 graph. Collect and Colour phases are stacked. (GraphLab – labelled GL, does not give distinct information; Collect-Hash is not visible as it is too fast.) The number of colours used is presented in boxes. The time for the final (right-most) GraphLab run was 919s, about three times the size of the bar shown.

Вычислительные эксперименты. Выводы

- На тестовых графах использование хэш-функции в качестве приоритета позволило сократить время работы в 2.5 – 3 раза за счет фазы сбора приоритетов.
- Использование приоритета LDF позволяет получить раскраску в меньшее число цветов (разница от 5 до 75%).
- Производительность ограничена затратами на коммуникацию: чем больше порядок графа, тем больше ребер, не локальных для процессов и больше время работы

Подход 2. Алгоритм для систем с общей памятью Чаталюрека (2012)

1. Разделить вершины на p равных блоков. V_i - множество вершин потока i.
2. Множество нераскрашенных вершин $U = V$
3. Пока есть нераскрашенные вершины $U \neq \emptyset$:
   1. *Предварительная раскраска*
      Параллельно На каждом потоке для всех $v \in V_i \cap U$:
      1. для каждой вершины v назначить минимальный доступный цвет
   2. Синхронизация потоков (барьер)
   3. Множество вершин, которые надо перекрасить, $R = \emptyset$.
   4. *Определение конфликтов*
      Параллельно На каждом потоке для всех $v \in V_i \cap U$:
      1. Если есть неправильно раскрашенное ребро $(w, v) \in E: c(v) = c(w), v > \mathrm{w}$, то $R = R \cup \{v\}$
   5. Синхронизация потоков (барьер)
   6. $U = R$

Подход 2. Алгоритм для систем с общей памятью Чаталюрека (2012)

Algorithm 2 The parallel graph coloring algorithm by Çatalyürek *et al.*.

Input: $\mathcal{G}(V, E)$
$\mathcal{U} \leftarrow V$
while $\mathcal{U} \neq \emptyset$ **do**
  #pragma omp parallel for ▷ Phase 1 - Tentative coloring (in parallel)
  for all vertices $V_i \in \mathcal{U}$ **do** ▷ execute First-Fit
    $\mathcal{C} \leftarrow$ {colors of all colored vertices $V_j \in adj(V_i)$}
    $c(V_i) \leftarrow$ {smallest color $\notin \mathcal{C}$}
  #pragma omp barrier
  $\mathcal{L} \leftarrow \emptyset$ ▷ global list of defectively colored vertices
  #pragma omp parallel for ▷ Phase 2 - Conflict detection (in parallel)
  for all vertices $V_i \in \mathcal{U}$ **do**
    if $\exists V_j \in adj(V_i), V_j > V_i : c(V_j) == c(V_i)$ **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup V_i$ ▷ mark V_i as defectively colored
  #pragma omp barrier
  $\mathcal{U} \leftarrow \mathcal{L}$ ▷ Vertices to be re-colored in the next round

Rokos G., Gorman G., Kelly P. H. J. A fast and scalable graph coloring algorithm for multi-core and many-core architectures //European Conference on Parallel Processing. – Springer, Berlin, Heidelberg, 2015. – С. 414-425.

Подход 2. Модификация алгоритма (Рокос и др., 2015)

Цель оптимизации: уменьшить синхронизацию между потоками

1. Разделить вершины на p равных блоков. V_i - множество вершин потока i.
2. *Предварительная раскраска*
   Параллельно На каждом потоке i для всех $v \in V_i \cap U$:
      1. для каждой вершины v назначить минимальный доступный цвет $c(v)$
3. Синхронизация потоков (барьер)
4. *Определение конфликтов*
   1. Множество вершин, которые надо проверить, $R_0 = V$. $k = 1$
   2. Пока $R_{k-1} \neq \emptyset$ (есть вершины, изменившие цвет на предыдущей итерации):
      1. Множество неправильно раскрашенных вершин $L = \emptyset$
      2. Параллельно На каждом потоке i для всех $v \in V_i \cap R_{k-1}$:
         Если есть неправильно раскрашенное ребро $(w, v) \in E: c(v) = c(w), w > v,$ то
         1. Назначить минимальный доступный цвет для v, $c(v)$
         2. $L = L \cup \{v\}$
      3. Синхронизация потоков (барьер)
      4. $R_k = L$; $k = k + 1$

Подход 2. Модификация алгоритма (Рокос и др., 2015)

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**
    $\mathcal{C} \leftarrow$ {colors of all colored vertices $V_j \in adj(V_i)$}
    $c(V_i) \leftarrow$ {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
            $\mathcal{C} \leftarrow$ {colors of all colored $V_j \in adj(V_i)$} ▷ re-color them
            $c(V_i) \leftarrow$ {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

Rokos G., Gorman G., Kelly P. H. J. A fast and scalable graph coloring algorithm for multi-core and many-core architectures //European Conference on Parallel Processing. – Springer, Berlin, Heidelberg, 2015. – C. 414-425.

Вычислительные эксперименты

- Rokos G., Gorman G., Kelly P. H. J. A fast and scalable graph coloring algorithm for multi-core and many-core architectures //European Conference on Parallel Processing. – Springer, Berlin, Heidelberg, 2015. – С. 414-425.

- Инфраструктура: a dual-socket Intel®Xeon® E5- 2650 system (Sandy Bridge, 2.00GHz, 8 physical cores per socket, 2-way hyperthreading) running Red Hat Enterprise Linux Server v. 6.4 (Santiago). Компилятор Intel®Composer XE 2013 SP1.
- Тестовые графы: 2D и 3D сетки порядка 220-250 * 10^3 вершин; графы RMAT трех типов с 16*10^6 вершин, 128*10^6 ребер.

Вычислительные эксперименты

- Сравнение времени работы оптимизированного алгоритма и алгоритма Чаталюрека

		Intel® Xeon®					
		Number of OpenMP threads					
		1	2	4	8	16	32
mesh2d	C:	62.7	34.0	19.2	10.2	5.92	4.28
	R:	62.2	31.3	17.7	9.42	5.50	4.05
bmw3_2	C:	58.1	33.5	14.4	7.84	4.73	3.61
	R:	57.8	29.4	12.1	6.48	3.91	3.30
pwtk	C:	40.1	24.0	14.5	8.07	4.96	3.65
	R:	39.8	20.0	11.3	6.08	3.81	3.30
RMAT-ER	C:	6.11	3.21	1.82	1.09	0.79	0.85
	R:	6.09	3.20	1.81	1.08	0.78	0.85
RMAT-G	C:	6.10	3.18	1.82	1.08	0.77	0.81
	R:	6.07	3.17	1.81	1.07	0.77	0.81
RMAT-B	C:	5.47	2.86	1.62	0.93	0.65	0.64
	R:	5.46	2.83	1.60	0.92	0.64	0.63

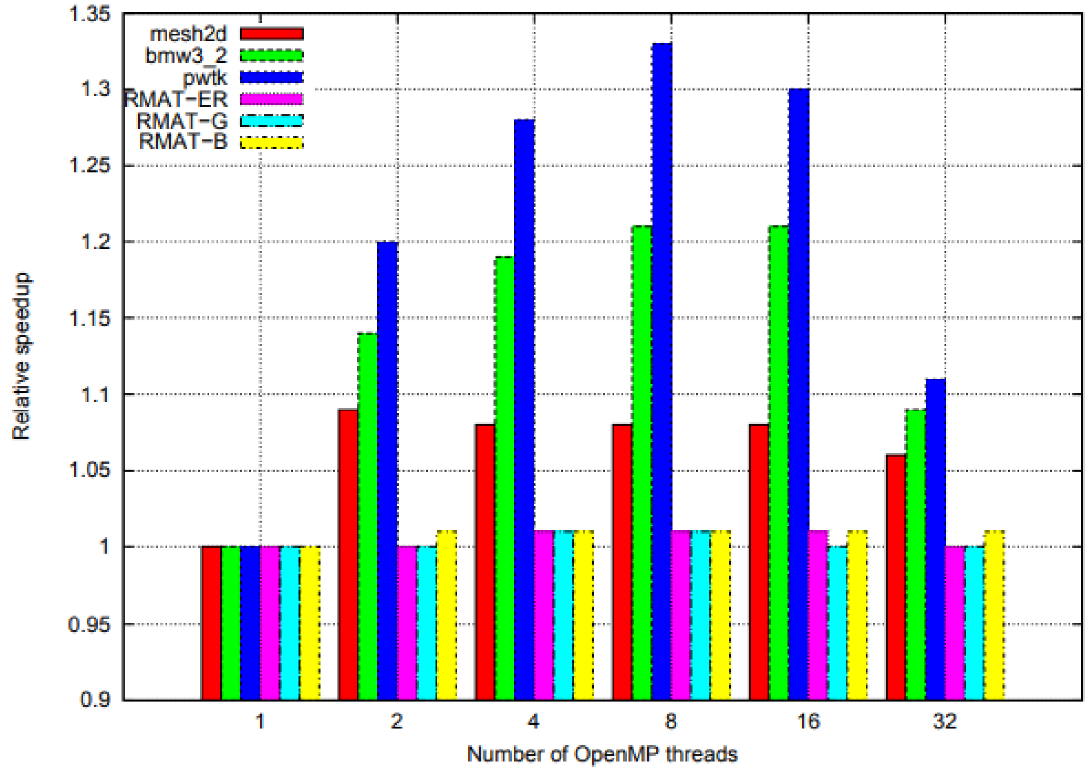
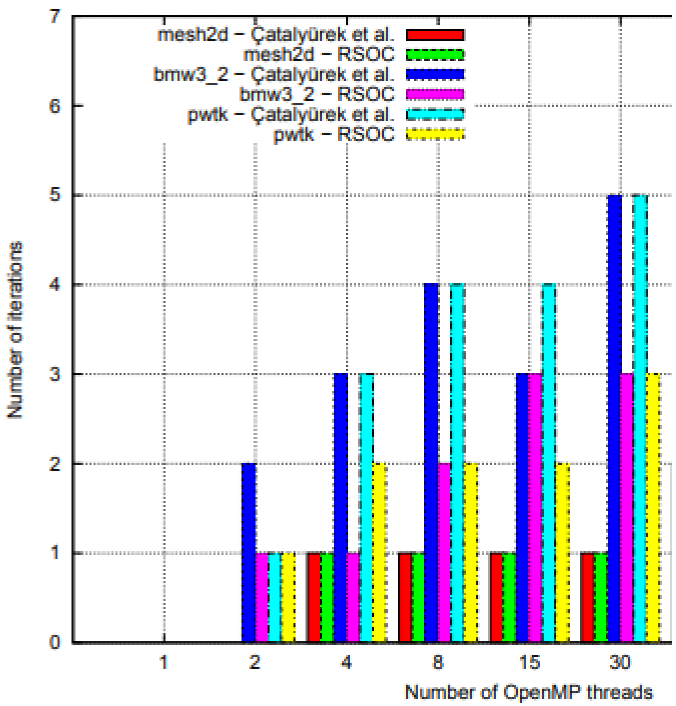


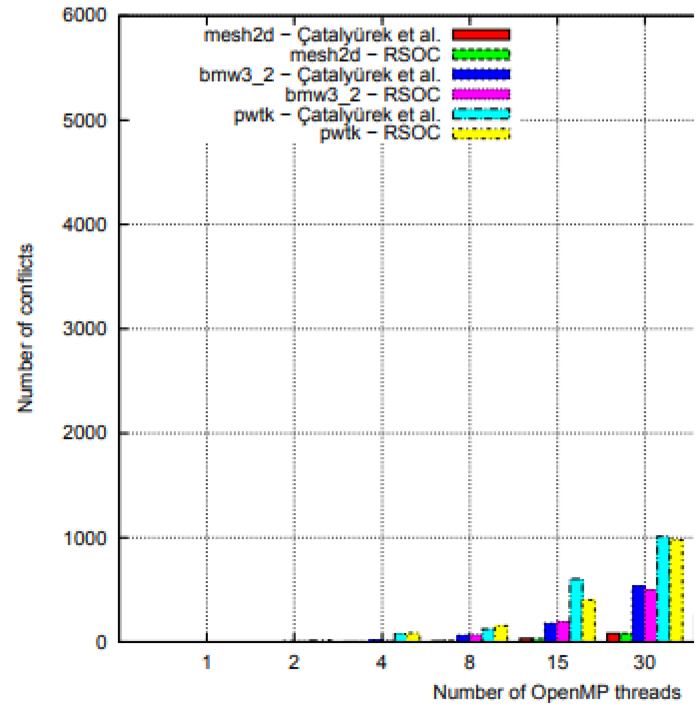
Fig. 1. Speedup of RSOC relative to Çatalyürek *et al.* as the number of threads increases on Intel® Xeon® E5-2650.

Вычислительные эксперименты

- Сравнение числа конфликтов и числа итераций алгоритмов



Число итераций для графов-сеток



Число конфликтов для графов-сеток

Вычислительные эксперименты. Выводы

- Для сеток алгоритм Рокоса показывает ускорение до 17 раз на 32 потоках, алгоритм Чаталюрека – до 16 раз. Для графов RMAT ускорение до 7,5 раз в среднем у обоих алгоритмов.
- Оптимизация барьерной синхронизации позволила сократить время работы на графах-сетках на 9-33% при работе в 2-16 потоков. На графах RMAT показано близкое время работы обоих алгоритмов.
- В оптимизированном алгоритме для графов-сеток число выполненных итераций сокращено на 1-2.
- Число конфликтов раскраски для обоих алгоритмов близко при использовании до 32 потоков.
- Число цветов, использованное алгоритмами, одинаково.

Подход 2. Алгоритм Боумана и др. (2005) для систем с распределенной памятью

1. Начальное распределение вершин между процессами. V_i - множество локальных вершин процесса i, всего p процессов
2. На каждом процессе P_i, $i = \overline{1, p}$:
   1. Множество нераскрашенных вершин $U_i = V_i$
   2. Пока есть процесс с нераскрашенными вершинами $U_i \neq \emptyset$:
      1. Разделить U_i на l_i подмножеств U_{ij} размера s
      2. *Супершаг.* Для каждого подмножества U_{ij}, $j = \overline{1, l_i}$:
         1. Для каждой вершины $v \in U_{ij}$ назначить доступный цвет
         2. Отправить цвета граничных вершин из U_{ij} соответствующим процессам
         3. Получить информацию с других процессов
      3. Множество вершин, которые надо перекрасить, $R_i = \emptyset$
      4. Для каждой граничной вершины $v \in U_i$:
         1. Если есть неправильно раскрашенное ребро $(w, v) \in E$: $c(v) = c(w), r(v) \leq r(w)$, где $r(v)$ - случайное число, то $R_i = R_i \cup \{v\}$
      5. $U_i = R_i$

Подход 2. Алгоритм Боумана и др. (2005) для систем с распределенной памятью

- Алгоритм разделен на супершаги для уменьшения коммуникаций между процессами. Чем меньше супершагов, тем больше вероятность ошибок раскраски, а значит, больше число внешних итераций алгоритма.
- Супершаги можно выполнять синхронно или асинхронно.
  - При синхронной работе цвет вершины проверяется только по соседям, которые получили цвет на этом же супершаге.
  - При асинхронной работе возможно больше конфликтов раскраски, так как у разных процессов может пересечься несколько супершагов. Цвет вершины проверяется по всем ее соседям.
- Выбор параметра s, при котором минимально общее время работы, зависит от плотности и размеров исходного графа, а также характеристик используемой вычислительной системы.

Подход 2. Алгоритм Боумана и др. (2005) для систем с распределенной памятью

- Выбор цвета вершины
  - First Fit: каждый процесс выбирает минимальный доступный цвет из $[1, C]$, где C – максимальный использованный цвет. Если такого цвета нет, то новый цвет $C + 1$.
  - Staggered First Fit: Пусть K – оценка числа цветов графа. Тогда каждый процесс P_i выбирает минимальный доступный цвет из $\left[\left\lceil\frac{iK}{p}\right\rceil, K\right]$. Если такого цвета нет, то минимальный доступный цвет из $\left[1, \left\lfloor\frac{iK}{p}\right\rfloor\right]$.
- Для сокращения числа коммуникаций при начальном разделении графа между процессами можно использовать алгоритмы разделения графа

Библиотеки

- ColPack – реализация последовательных алгоритмов https://cscapes.cs.purdue.edu/coloringpage/
- Zoltan – реализация алгоритма Боумана (MPI) http://www.cs.sandia.gov/Zoltan/ug_html/ug_color_parallel.html
- Zoltan2 – реализация последовательных алгоритмов
- KokkosKernels – распараллеливание алгоритмов раскраски в рамках одного вычислительного узла с использованием библиотеки Kokkos (CPU, GPU, KNL) https://github.com/kokkos/kokkos-kernels
- В составе библиотек:
  - Parallel Boost Library – реализация алгоритма Боумана
  - cuSparse – https://devblogs.nvidia.com/graph-coloring-more-parallelism-for-incomplete-lu-factorization/

Заключение

- Алгоритмы раскраски графа широко используются для распараллеливания других алгоритмов на графах, алгоритмов разреженной алгебры, в задачах составления расписания.
- Последовательный алгоритм основан на жадной стратегии назначения цветов, при этом вершины графа посещаются в порядке приоритета. От порядка обхода вершин зависит качество раскраски и время работы алгоритма.

Заключение

- Два типа параллельных алгоритмов: алгоритмы, основанные на раскраске независимого множества вершин и итерационные алгоритмы, основанные на спекулятивной раскраске вершин с последующим исправление конфликтов.
- Параллельные алгоритмы можно реализовать в синхронном и асинхронном варианте
- Модификации базовых параллельных алгоритмов направлены на сокращение коммуникаций и синхронизаций между потоками (процессами).

Литература

1. Hasenplaugh W. et al. Ordering heuristics for parallel graph coloring //Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures. – ACM, 2014. – P. 166-177.
2. Çatalyürek Ü.V. et al. Graph coloring algorithms for multi-core and massively multithreaded architectures //Parallel Computing. – 2012. – Т. 38. – №. 10-11. – С. 576-594.
3. Rokos G., Gorman G., Kelly P. H. J. A fast and scalable graph coloring algorithm for multi-core and many-core architectures //European Conference on Parallel Processing. – Springer, Berlin, Heidelberg, 2015. – С. 414-425.
4. Boman E. G. et al. A scalable parallel graph coloring algorithm for distributed memory computers //European Conference on Parallel Processing. – Springer, Berlin, Heidelberg, 2005. – С. 241-251.
5. Sallinen S. et al. Graph colouring as a challenge problem for dynamic graph processing on distributed systems //SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. – IEEE, 2016. – С. 347-358.

Литература

Дополнительные алгоритмы:

1. Deveci M. et al. Parallel graph coloring for manycore architectures //2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS). – IEEE, 2016. – С. 892-901.
2. Firoz J. S., Zalewski M., Lumsdaine A. A Synchronization-Avoiding Distance-1 Grundy Coloring Algorithm for Power-Law Graphs //2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT). – IEEE, 2019. – С. 421-432.
3. Naumov M., Castonguay P., Cohen J. Parallel graph coloring with applications to the incomplete-LU factorization on the GPU //Nvidia White Paper. – 2015.

Контакты

Нижегородский государственный университет

http://www.unn.ru

Институт информационных технологий, математики и механики

http://www.itmm.unn.ru

Пирова А.Ю.

anna.pirova@itmm.unn.ru