



دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

گزارش پروژه درس طراحی الگوریتم

رنگ آمیزی گراف به صورت موازی

دانشگاه صنعتی امیرکبیر، دانشکده مهندسی کامپیوتر و فناوری اطلاعات

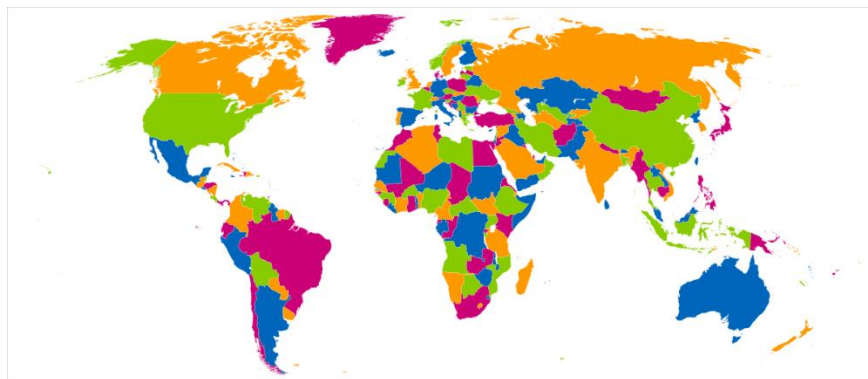
پارسا اسکندرنازاد - 9531003

فهرست

1.....	1. مقدمه
2.....	2. روش های ترتیبی
3.....	First Fit
3.....	Largest-Degree-Ordering
4.....	3. روش های موازی
4.....	Maximal Independent Set
5.....	Jones – Plassmann
5.....	Parallelized First Fit
7.....	Block Partition Based
8.....	4. پیاده سازی
9.....	5. صحت عملکرد و بررسی نتایج
12.....	6. منابع

1. مقدمه

مسئله رنگ‌آمیزی (راس‌های) گرافⁱ به مسئله‌ای گفته می‌شود که در آن به هر راس از یک گراف یک برچسبⁱⁱ به عنوان رنگ نسبت داده می‌شود به طوری که هیچ دو راس مجاوری رنگ مشابه نداشته باشند.



رنگ‌آمیزی نقشه جهان توسط چهار رنگ

این مسئله در اثر تلاش برای رنگ‌آمیزی نقشه‌های جغرافیایی مطرح شد. فرانسیس گوتریهⁱⁱⁱ قضیه چهار رنگ را در تلاش برای رنگ‌آمیزی نقشه انگلستان مطرح کرد و نشان داد که تنها به چهار رنگ نیاز است تا همه مناطق رنگ‌آمیزی

شوند بدون این که هیچ دو منطقه اطراف هم رنگ یکسان داشته باشند.^[1]

روش‌های متنوع و زیادی برای رنگ‌آمیزی راس‌های گراف موجود است که نتیجه به دست آمده بنا به روش، ممکن است متفاوت باشد. به حداقل رنگ‌های مورد نیاز برای رنگ‌آمیزی گراف عدد کروماتیک^{iv} آن گفته می‌شود.

از رنگ‌آمیزی گراف در کاربردهای متعددی نظیر مسائل زمان‌بندی^v، تخصیص منابع در سخت‌افزار، حل جدول سودوکو، پیدا کردن الگو و ... استفاده می‌شود.

تا کنون الگوریتم‌های ترتیبی^{vi} زیادی برای این مسئله پیشنهاد شده است که در گراف‌های با ابعاد کوچک بسیار خوب عمل می‌کند اما در مقیاس‌های بزرگتر ممکن است که با کاهش کارایی مواجه شوند. لذا به نظر می‌رسد بررسی روش‌های موازی^{vii} بتواند کارایی را بهبود ببخشد.

ⁱ Graph Coloring Problem (GCP)

ⁱⁱ Label

ⁱⁱⁱ Francis Guthrie

^{iv} chromatic number

^v Scheduling

^{vi} Sequential

^{vii} Parallel

همچنین در زمینه حل این مسئله به صورت موازی نسبت به راه‌حل‌های ترتیبی نیز به مراتب، تحقیقات و مطالعات بسیار کمتری شده است.^[2]

در این مطلب ابتدا به صورت خلاصه به این روش‌ها می‌پردازیم و سپس یک پیاده‌سازی از یکی از این الگوریتم‌ها را به زبان جاوا ارائه می‌دهیم.

2. روش‌های ترتیبی

به منظور درک بهتر و داشتن تصویری از راه‌حل‌های موازی ابتدا بهتر است روش‌های ترتیبی را بررسی کنیم.

همانطور که اشاره شد، تا کنون تلاش‌های بسیاری برای حل این مسئله به صورت ترتیبی شده است که نتیجه بسیاری از این الگوریتم‌ها نظیر (LDO) Largest-Degree- Ordering، Incidence-Degree- Ordering (IDO)، First Fit (FF) و ... ارائه رنگ‌آمیزی‌هایی معقول و کارا بوده است.^[2]

در ادامه چند نمونه را بررسی خواهیم کرد.

یک از روش‌های معقولی که پیاده‌سازی آن نیز ساده است روش Greedy می‌باشد. در این روش در هر زمان انتخابی که به نظر بهترین است انجام می‌شود. محاسبات نشان داده‌اند که نتایج حاصل از این الگوریتم‌ها با تقریب بسیار خوبی نزدیک به نتیجه بهینه است.^[2]

1-First Fit

الگوریتم First Fit (FF) یکی از الگوریتم‌های Greedy است که به سرعت بالا مشهور است. این الگوریتم به صورت ترتیبی راس‌ها را پیمایش کرده و به هر راس کمترین برچسب ممکن را اختصاص می‌دهد.

```

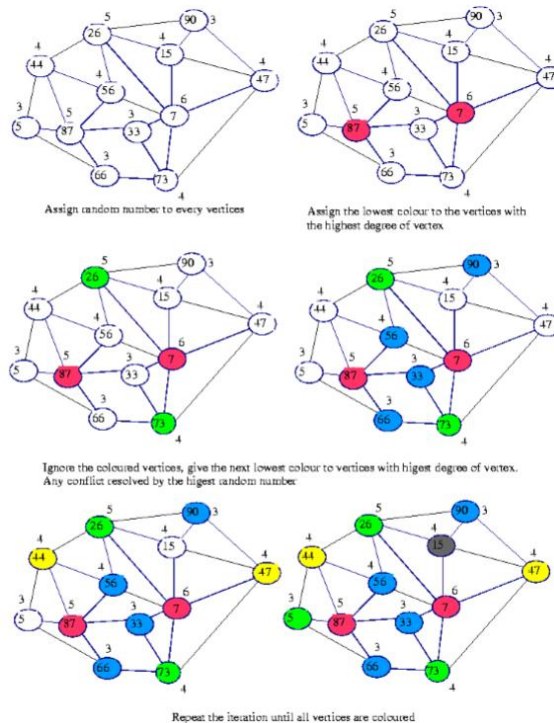
FirstFit(G)
begin
  for  $i = 1$  to  $n$  do
    assign smallest legal color to  $v_i$ 
  end-for
end

```

الگوریتم First Fit^[3]

2-Largest-Degree-Ordering

این الگوریتم ابتدا به هر راس عددی را نسب می‌دهد که برابر با درجه آن راس می‌باشد. سپس به صورت نزولی شروع کرده و رنگ‌آمیزی را انجام می‌دهد. LDO رنگ‌آمیزی را بهتر از FF انجام می‌دهد چرا که در هر مرحله ابتدا راسی رنگ می‌شود که همسایه‌های بیشتری دارد که در نتیجه شانس انتخاب رنگ بزرگتر برای آن بیشتر است.^[2] [3]



رنگ‌آمیزی یک گراف با LDO^[2]

3. روش‌های موازی

در حقیقت به احتمال بالا، رنگ‌آمیزی گراف فقط بخشی از حل یک مسئله بزرگ‌تر است و در صورت بزرگ بودن اطلاعات در حال پردازش، ممکن است زمان و هزینه سخت‌افزاری بالایی صرف انجام رنگ‌آمیزی شود. در روش‌های موازی هرچند شاید کیفیت رنگ‌آمیزی کمی با روش‌های ترتیبی فرق داشته باشد اما از نظر سرعت بهبود چشمگیری را خواهیم داشت. همچنین بهتر است این نکته را در نظر بگیریم که در خیلی از کاربردها برای تعداد رنگ‌ها محدودیتی وجود ندارد و شاید حل مسئله در زمان کمتر مفیدتر باشد. مطالعات و منابع در مورد روش‌های موازی بسیار محدود هستند و اکثر آن‌ها نیز تلاش کرده اند تا روش‌های ترتیبی را موازی کنند.^[2]

ارائه روش و پیاده‌سازی روش‌های موازی مستلزم این است که بدانیم چه کارهایی قابل موازی‌سازی هستند و چگونه این کارها را انجام دهیم. برای بررسی این روش‌ها لازم است بدانیم که پایه و اساس اکثر این روش‌ها طبق این واقعیت است که رنگ‌آمیزی گراف را می‌توان به این صورت انجام داد که هر زیرمجموعه مستقل از گراف (زیر مجموعه‌ای که هیچ دو راسی همسایه نباشند) را به صورت موازی رنگ کنیم.^[4]

```

U := V
while (|U| > 0) do in parallel
    Choose an independent set I from U
    Color all vertices in I
    U := U - I
end do

```

منطق اکثر الگوریتم‌های موازی رنگ‌آمیزی^[4]

در ادامه چند نمونه را بررسی خواهیم کرد و سپس آن‌ها را نقد می‌کنیم. و در آخر منطق دیگری را معرفی می‌کنیم.

1. Maximal Independent Set

در این الگوریتم در هر دفعه یک Maximal Independent Set (MIS) پیدا شده و به همه اعضای آن مجموعه یک رنگ اختصاص داده می‌شود و سپس این زیرمجموعه از گراف حذف می‌شود. Luby در مقاله‌ای که ارائه داد قسمت ساخت MIS در هر مرحله را طوری تغییر داد که به طور موازی انجام شود.

[4]

2. Jones – Plassmann

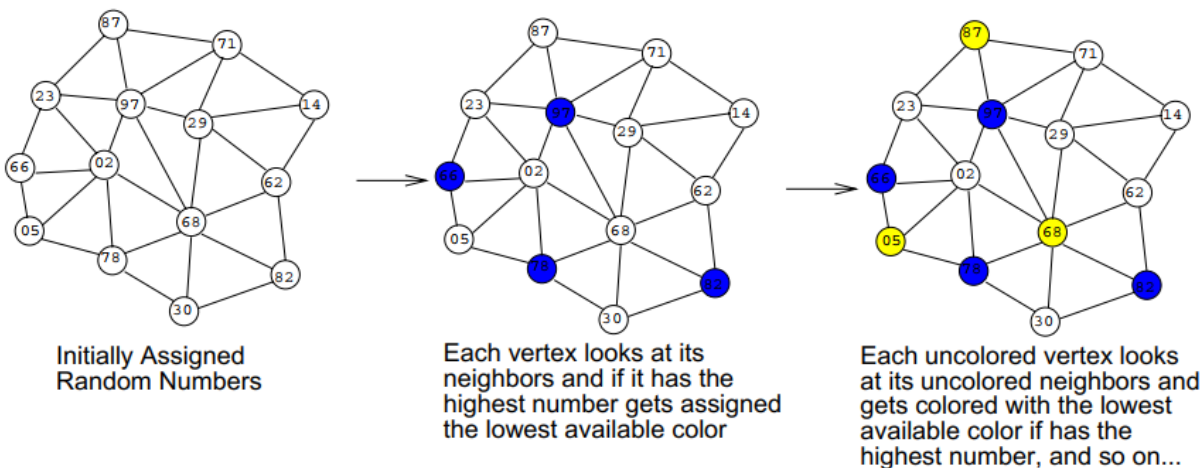
این روش در حقیقت بهبودیافته روش MIS است. ابتدا به هر راس یک عدد تصادفی اختصاص داده می‌شود و سپس بررسی می‌شود که آیا همسایه‌ها عدد بزرگتری نداشته باشند. این روش بررسی یک مجموعه مستقل در اختیار ما قرار می‌دهد که قادر خواهیم بود همه اعضای آن را به صورت موازی رنگ کنیم.^[4]

```

U := V
while (|U| > 0) do
  for all vertices v ∈ U do in parallel
    I := {v such that w(v) > w(u) ∀ neighbors u ∈ U}
    for all vertices v' ∈ I do in parallel
      S := {colors of all neighbors of v'}
      c(v') := minimum color not in S
    end do
  end do
  U := U - I
end do

```

الگوریتم Jones – Plassmann^[4]



یک رنگ آمیزی با JP^[4]

از معایب این الگوریتم نیز می‌توان به بهینه نبودن رنگ‌بندی و همچنین balance نبودن آن اشاره کرد.^[2] روش‌های دیگری نیز بر مبنای این منطق وجود دارد که به جهت طولانی نشدن مطلب از توضیح آن‌ها صرف نظر می‌کنیم.

3. Parallelized First Fit

به جهت درک بهتر نسخه موازی شده این الگوریتم لازم است که ابتدا نسخه ترتیبی آن را بررسی کنیم.

نسخه ترتیبی دو پایه اساسی دارد که به این شرح است^[5]:

1. ساخت لیستی از رنگ‌های مجاز با توجه به رنگ‌های همسایه‌ها:

$\text{Build}(L_i, v_j)$

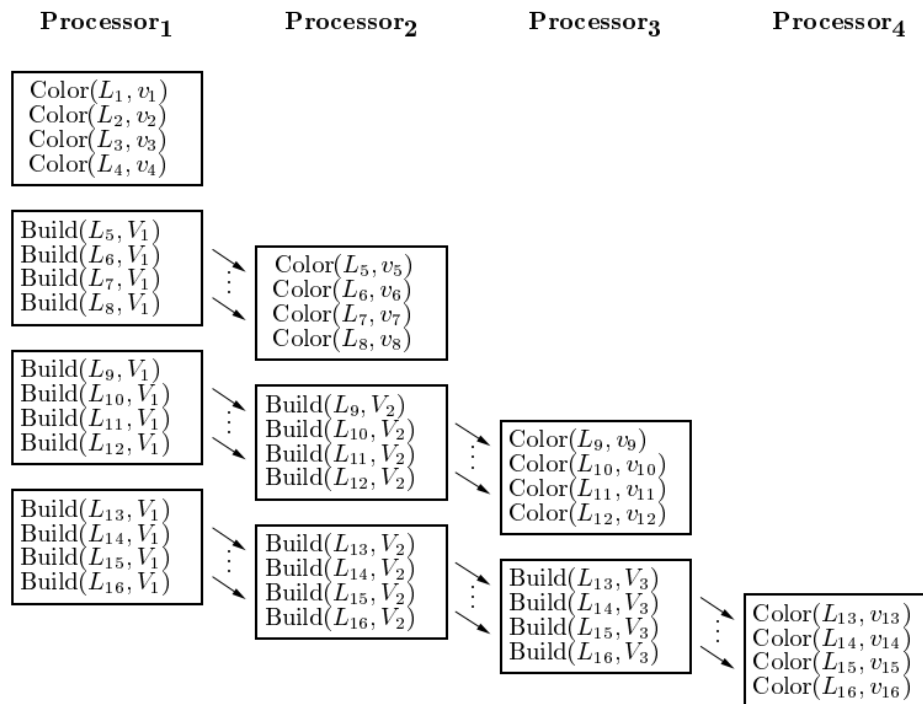
رنگ راس V_j را از لیست رنگ‌های مجاز V_i حذف می‌کند.

2. رنگ‌آمیزی با توجه به لیست ارائه شده در مرحله قبل:

$\text{Color}(L_i, v_i)$

راس V_i را با توجه به لیست رنگ‌های مجازش رنگ می‌کند.

انجام چند مرحله از این الگوریتم به صورت موازی امکان پذیر است که نحوه انجام آن در شکل‌های زیر نشان داده شده است.



اجرای موازی الگوریتم^[6] FF

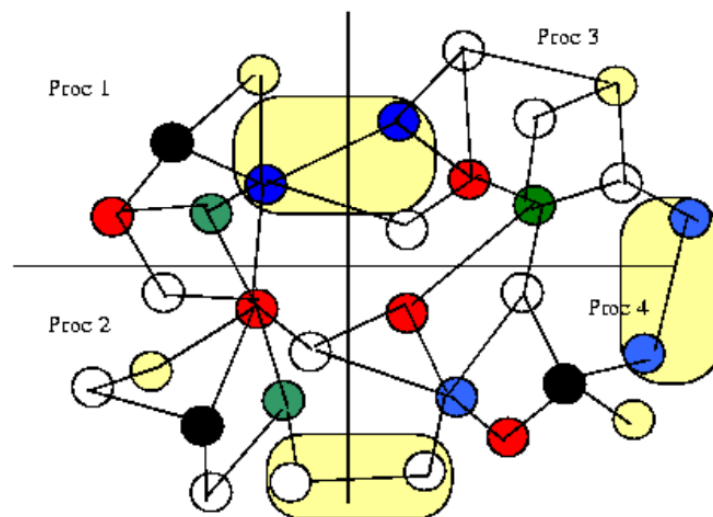
همانطور که اشاره شد هدف از ارائه این الگوریتم‌ها افزایش سرعت بوده است، اما تعداد زیادی از این الگوریتم‌ها در یک آزمایش رقابتی مورد آزمایش قرار گرفتند و در نهایت افزایش سرعت چشمگیری مشاهده نشد. همچنین Jones و Plassmann نیز در مقاله خود به افزایش سرعت اشاره نکردند.^[6]

در ادامه به یک الگوریتم جدید که پیاده‌سازی آن نیز انجام گرفته است اشاره می‌کنیم.

4. Block Partition Based

این روش که اسم‌های دیگر آن Parallel First Fit (توجه شود که با الگوریتم قبلی کمی متفاوت است.) و یا Gebremedhin - Manne می‌باشد دارای سه فاز اصلی است. در فاز اول به جای یافتن مجموعه راس مستقل، گراف را تقسیم کرده و تعداد مساوی راس را بدون توجه به ارتباط آن‌ها به پردازنده‌ها اختصاص می‌دهیم. به این کار به اصطلاح پارتیشن کردن گفته می‌شود.

در طی انجام عملیات موازی رنگ کردن هر پارتیشن توسط هر پردازنده، ممکن است دو همسایه مربوط به دو پارتیشن جدا، همزمان رنگ شوند و در رنگ‌آمیزی خطا به وجود بیاید. در نتیجه به رنگ‌آمیزی انجام شده در این فاز شبه رنگ‌آمیزی می‌گوییم.



یک شبه رنگ‌آمیزی که خطاهای آن مشخص شده است.^[2]

در فاز بعدی توسط هر پردازنده به طور موازی خطاهای رنگ‌آمیزی هر قسمت را پیدا کرده و در جایی نگه‌داری می‌کنیم. توجه شود که هدف این کار اصلاح آن‌ها در فاز بعدی است لذا کافیهست از جفت راس‌های مشترک در یک یالی که باعث خطا شده‌اند فقط یکی از آن‌ها را ذخیره کنیم. (مثلا کوچک‌ترین شماره اندیس آن‌ها را)

در فاز آخر خطاهای موجود را به صورت ترتیبی اصلاح می‌کنیم.

در این روش باید توجه داشته باشیم که با توجه امکان رنگ‌آمیزی غلط یا درست در مرزها ممکن است برای یک گراف چند رنگ‌آمیزی مختلف ولی درست و نزدیک به بهینه داشته باشیم.

BlockPartitionBasedColoring(G, p)

begin

1. Partition V into p equal blocks $V_1 \dots V_p$, where $\left\lfloor \frac{n}{p} \right\rfloor \leq |V_i| \leq \left\lceil \frac{n}{p} \right\rceil$

for $i = 1$ to p do in parallel

for each $v_j \in V_i$ do

assign the smallest legal color to vertex v_j

barrier synchronize

end-for

end-for

2. for $i = 1$ to p do in parallel

for each $v_j \in V_i$ do

for each neighbor u of v_j that is colored at the same parallel step do

if $color(v_j) = color(u)$ then

store $\min\{u, v_j\}$ in table A

end-if

end-for

end-for

end-for

3. Color the vertices in A sequentially

end

^[6] الگوریتم Block Partition Based

4. پیاده‌سازی

برای پیاده‌سازی الگوریتم Parallel First Fit (Gebremedhin – Manne) در این پروژه از زبان جاوا استفاده شد.

برای انجام کارهای موازی از Threadهای جاوا استفاده شد که اجازه کنترل و پیاده‌سازی اعمال موازی را می‌دهد.

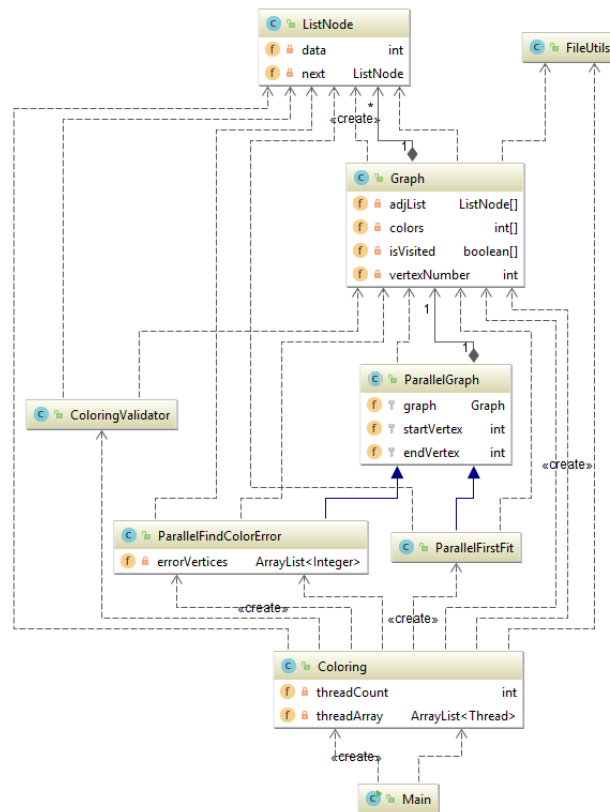
برای استفاده از Thread دو راه اصلی وجود دارد:

1. Extend کردن کلاس Thread و Override کردن متد run

2. Implement کردن اینترفیس Runnable

ما در این پروژه از روش دوم استفاده می‌کنیم که همزمان بتوانیم کلاس دیگری را نیز extend کنیم.

شکل زیر نمودار نحوه ارتباط کلاس‌ها را نشان می‌دهد:



Powered by yFiles

همانطور که قبل تر اشاره شد الگوریتم پیاده سازی شده به این شرح است:

تقسیم گراف به مجموعه های هم اندازه و اختصاص دادن هر مجموعه به یک پردازنده (اینجا thread)

1. رنگ آمیزی موازی هر بخش مشابه الگوریتم FF

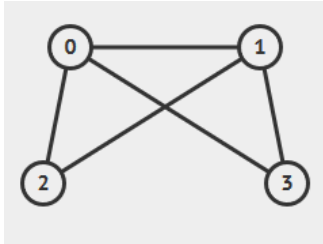
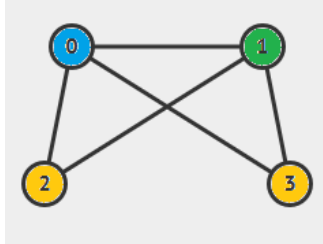
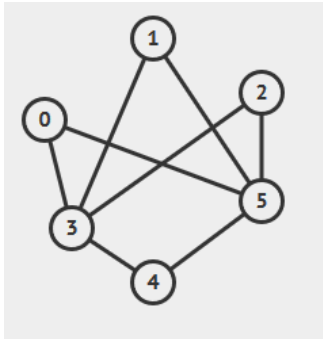
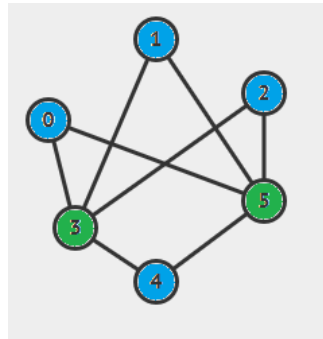
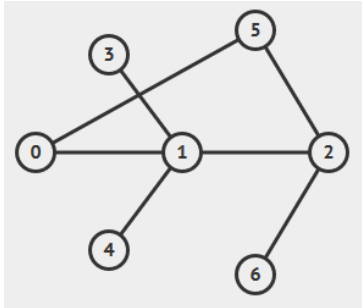
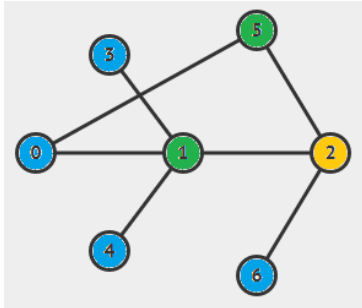
2. پیدا کردن موازی خطاها در شبه رنگ آمیزی انجام شده در هر بخش

3. برطرف کردن خطاها به صورت ترتیبی

5. صحت عملکرد و بررسی نتایج

ابتدا نتایج حاصل از رنگ آمیزی چند گراف را بررسی می کنیم.

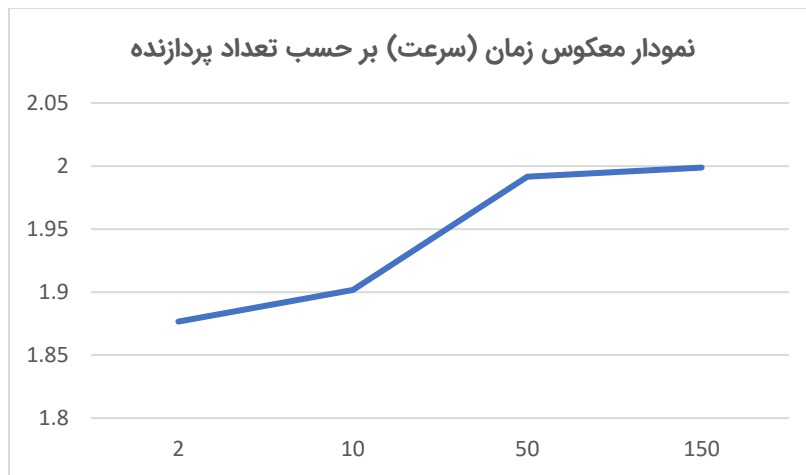
* توجه شود همانطور که قبل تر هم اشاره شد با توجه به احتمالاتی بودن این که چند thread هم زمان راس های یک یال را رنگ کنند یا نه ممکن است نتایج مختلفی برای یک گراف داشته باشیم که البته همه آنها درست و نزدیک به بهینه می باشند.

نتایج اجرای الگوریتم رنگ‌آمیزی توسط 2 پردازنده (thread) به طور موازی	
ورودی	خروجی
	
	
	

مثال‌های بالا مثال‌های کوچکی بودند که به جهت بررسی صحت و قابلیت تایید آن با نگاه آورده شده‌اند. همچنین باید در نظر داشته باشیم که برای گراف‌های کوچک، حتی ممکن است ساخت و راه‌اندازی threadها هزینه زمانی بیشتری نسبت به هزینه جبران شده توسط عملیات موازی تحمیل کنند و شاهد کاهش سرعت باشیم. برای بررسی عملکرد این الگوریتم بر روی گراف‌های بزرگ نتیجه یک آزمایش بر روی یک گراف با سایز 46 مگ و 150000 راس توسط کامپیوتری به مشخصات زیر را در ادامه می‌بینیم.

CPU Model	Intel Core i7-7700HQ Kaby Lake
Core	4
CPU L1 Cache	6144 Kb
RAM Capacity	16 Gb DDR4
OS	Windows 10- 64 bit

number of processors	time (ms)
2	53289
10	52587
50	50214
150	50031



* این آزمایش یک مشاهده ابتدایی است و برای داشتن داده دقیق، بهتر است آزمایش چندبار انجام و محاسبات آماری بر روی آن انجام شود.

پیچیدگی زمانی این الگوریتم $O(\Delta n/p)$ است که Δ نشان‌دهنده بیشترین درجه گراف است. اثبات این ادعا از این بحث خارج است و در صورت تمایل در مقاله ارائه شده پیدا می‌شود.^[6]

6. منابع

منابعی که به صورت مستقیم در متن به آن‌ها اشاره شده است:

- ¹ M. Kubale, History of graph coloring, in Kubale (2004)
- ² Parallel Graph Colouring Algorithms for Shared-Memory Machines, Ismet Isnaini, B.Eng. June 2002 Department of Computer Science The University Of Adelaide, South Australia
- ³ Parallel Graph Coloring By Assefaw Hadish Gebremedhi
- ⁴ A Comparison of Parallel Graph Coloring Algorithms, J. R. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer, C. L. Martin
- ⁵ Parallel Graph Coloring using JAVA, Thomas UMLAND, Deutsche Telekom Berkom GmbH, Goslarer Ufer 35, 10589 Berlin, Germany
- ⁶ Scalable parallel graph coloring algorithms, Assefaw Hadish Gebremedhin and Fredrik Manne, CONCURRENCY: PRACTICE AND EXPERIENCE, Department of Informatics, University of Bergen, N-5020 Bergen, Norway

منابعی که به صورت مستقیم در متن به آن‌ها اشاره نشده است ولی در فهم مطلب اثر داشته‌اند:

- A PARALLEL GRAPH COLORING HEURISTIC, Mark T. Jones and Paul E. Plassmann
- A SIMPLE PARALLEL ALGORITHM FOR THE MAXIMAL INDEPENDENT SET PROBLEM, MICHAEL LUBY
- Ordering Heuristics for Parallel Graph Coloring ,William Hasenplaugh Tim Kaler Tao B. Schardl Charles E. Leiserson ,MIT Computer Science and Artificial Intelligence Laboratory
- Parallel First-Fit Graph Coloring in Java ,Loutfouz Zaman ,Department of Computer Science and Engineering, York University