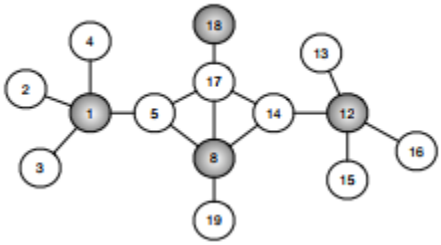
مجموعه مستقل ماکسیمال[1] سریع حافظه توزیع شده



شکل1) گره‌های خاکستری یک مجموعه مستقل ماکسیمال از این گراف را نشان می‌دهند.

چکیده

مسئله گراف مجموعه مستقل ماکسیمال (که به اختصار MIS نامیده می‌شود) در بسیاری از برنامه‌های کاربردی نظیر بینایی کامپیوتر[2]، نظریه اطلاعات[3]، زیست‌شناسی مولکولی[4] و زمانبندی ظاهر شده است. مقیاس در حال رشد مسائل MIS پیشنهاد استفاده از سخت‌افزار حافظه توزیع‌شده را به عنوان روشی مقرون به صرفه برای ارائه محاسبات لازم و منابع حافظه را می‌دهد. لوبی[5] چهار الگوریتم تصادفی برای حل مسئله MIS ارائه کرده است. تمامی این الگوریتم‌ها با تمرکز بر دستگاه‌های حافظه مشترک طراحی شده‌اند و با استفاده از مدل PRAMمورد تجزیه و تحلیل قرار گرفته‌اند. این الگوریتم‌ها دارای پیاده‌سازی‌های کارآمد مستقیم حافظه توزیع‌شده نیستند. در این مقاله، ما دو مورد از الگوریتم‌های MIS بدوی لوبی را برای اجرای حافظه توسعه‌یافته گسترش خواهیم داد که نام

[1] Maximal ،عضو بیشین نیز نامیده می‌شود

[2] computer vision
[3] information theory
[4] molecular biology
[5] Luby

آن‌ها Luby (A)وLuby(B) است و عملکرد آن‌ها را ارزیابی می‌کنیم. ما نتایج خود را با پیاده‌سازی «MISفیلترشده[6]» در کتابخانه Combinatorial BLASبرای دو نوع ورودی از گراف‌های مصنوعی مقایسه کردیم.

1. مقدمه

G = (V, E) را به عنوان گرافی در نظر بگیرید که در آن V نشان‌دهنده مجموعه‌ای از راس‌ها و E نشان‌دهنده مجموعه‌ای از یال‌ها است. یک مجموعه مستقل درGمجموعه‌ای از رئوسی در یک گراف است که هیچ دو راسی در مجموعه، مجاور نیستند. بزرگترین مجموعه‌های مستقل (که ممکن است بیش از یک باشد) ماکسیمم مجموعه‌های مستقل نامیده می‌شود. بنابراین یافتن ماکسیمم مجموعه مستقل ان‌پی-سخت[7] است، اکثر برنامه‌های کاربردی برای یافتن یک مجموعه مستقل ماکسیمال تنظیم شده‌اند. یک مجموعه مستقل ماکسیمال (MIS) از یک گراف یک مجموعه مستقل است که زیرمجموعه‌ای از مجموعه مستقل دیگر نیست (شکل 1 را ببینید). یافتن یک MIS یک مسئله مهم گراف است زیرا در بسیاری از برنامه‌های کاربردی از جمله بینایی کامپیوتر، نظریه اطلاعات، زیست‌شناسی مولکولی و زمانبندی فرآیند ظاهر شده است. اگرچه الگوریتم‌های موثر MISشناخته شده‌اند[1]، مقیاس در حال افزایش برنامه‌های کاربردی حساس به داده پیشنهاد استفاده از سخت‌افزار حافظه توزیع‌شده (خوشه‌ها[8]) را می‌دهد، که به نوبه خود نیاز به الگوریتم‌های توزیع حافظه دارند.

از الگوریتم‌های MIS، لوبی مونت کارلو [2][9] برای پیاده‌سازی MIS به شکل موازی استفاده می‌شود. الگوریتم‌های MIS لوبی با تمرکز بر دستگاه‌های حافظه مشترک طراحی شده‌اند و با استفاده از مدل دستگاهی با دسترسی تصادفی موازی PRAMمورد تجزیه و تحلیل قرار گرفته‌اند. الگوریتم‌های لوبی بلافاصله خود را به الگوریتم‌های موازی توزیع شده موثر قرض نمی‌دهند زیرا ممکن است سربار در اثر هماهنگ‌سازی و محاسبات زیرگراف توزیع‌شده

[6] FilteredMIS
[7] NP-hard
[8] clusters
[9] Luby's Monte Carlo

رخ دهد. در این مقاله، نسخه‌های توزیع‌شده الگوریتم‌های لوبی مونت کارلو (الگوریتم A و B) ارائه شده است که سبب به حداقل‌رسانی این سربارها می‌شود. علاوه بر این، ما یک متغیر از Luby (A)را مشتق کرده‌ایم که از تکرار محاسبه اعداد تصادفی در هر تکرار جلوگیری می‌کند. تمامی الگوریتم‌ها در زمان اجرای AM++[3] پیاده‌سازی شده‌اند و عملکرد آن‌ها ارزیابی شده است. نتایج ما نشان می‌دهد که الگوریتم‌های پیشنهادی به خوبی در تنظیمات توزیع‌شده قرار می‌گیرد. ما همچنین نتایج خود را با پیاده‌سازی MISفیلترشده در کتابخانه Combinatorial BLASمقایسه کرده‌ایم [4] و نشان داده‌ایم که پیاده‌سازی‌های ما چندین برابر سریع‌تر از الگوریتم MISفیلترشده است.

2. کارهای مرتبط

اکثر الگوریتم‌های MIS موازی بر روی حافظه اشتراکی تمرکز داشتند، و از مدل PRAM برای تجزیه و تحلیل پیچیدگی موازی استفاده می‌کردند (برای مثال [5]،[6] ،[7] ،[8]، [9]، [10]). در این کار، ما به طور خاص بر روی الگوریتم‌های تصادفی لوبی [2] تمرکز کرده‌ایم (برای جزئیات بیشتر به بخش 3 مراجعه کنید). لوبی یک تحلیل دقیق از الگوریتم‌های خود را با استفاده از مدل دستگاهPRAM ارائه کرد. بعدها، مفاهیم الگوریتم لوبی به منظور پیاده سازی نسخه‌های توزیع‌شده مورد استفاده قرار گرفت. لینچ[10] و همکاران [11] در مورد یک نسخه توزیع‌شده از الگوریتم لوبی برای شبکه‌های توزیع همزمان را مورد بررسی قرار دادند. متی‌ویر[11] و همکاران [12] یک نسخه بهبودیافته از الگوریتم لینچ را ارائه دادند که در آن پیچیدگی پیام ارتباطی بهبود یافته بود. کوهن[12] و همکاران [13] یک الگوریتمMIS توزیع شده قطعی را ارائه دادند. با این حال، آن‌ها یک مدل ارتباطی همزمان را در نظر گرفتند و هیچگونه نتیجه تجربی ارائه نکردند.

کتابخانه Combinatorial BLAS[4]یک نسخه توزیع‌شده از الگوریتم لوبی را پیاده‌سازی کرده است. این پیاده‌سازی از عناصر اولیه جبر خطی در پیاده‌سازی الگوریتم لوبی استفاده می‌کرد و همچنین این الگوریتم بر روی

[10]Lynch
[11]Metivier
[12]Kuhn

گراف‌های فیلترشده نیز کار می‌کرد [14]. سالیهگلو[13] و همکاران [15] یک نسخه توزیع شده از الگوریتم لوبی را برای سیستم‌های Pregel مانند اجرا کردند. آن‌ها از الگوریتم MIS برای حل مشکل رنگ‌آمیزی گراف استفاده کردند. گاریملا[14] و همکاران [16] عملکرد پیاده‌سازی لوبی در Pregel را با یک الگوریتم موازی که توسط بللوچ[15] و همکاران [10] طراحی شده بود مقایسه کردند. بللوچ و همکاران، الگوریتم MIS با ترتیب الفبایی اول که دارای ترتیبی حریصانه بود را موازی کردند. این الگوریتم از DAGاولیتی بر روی راس‌های گراف ورودی استفاده می‌کردند که یال‌های DAF نقاط پایانی اولویت بالاتر را به نقاط پایانی اولیت پایین‌تر بر اساس مقادیر تصادفی تخصیص یافته به راس‌ها متصل می‌کرد.

بر روی مساله MIS موازی در بیشتر تحقیقات نظری تمرکز شده بود. بخشی از کار مرتبطی که در بالا مورد بحث واقع شد شامل تحلیل پیچیدگی زمانی موازی یا پیچیدگی کمی الگوریتم است. پیاده‌سازی‌های موجود از MIS موازی عمدتا از الگوریتم لوبی استفاده می‌کنند یا اینکه آن‌ها از الگوریتمی بر اساس MIS لوبی استفاده کرده‌اند. با این حال، MIS لوبی به سرعت به الگوریتم موازی توزیع حافظه موثر گسترش پیدا نمی‌کند دلایل این موضوع هم در بخش 3 مورد بحث قرار خواهد گرفت.

الگوریتم1) طرح تکرارشونده کلی در MIS لوبی

Input: Graph $G = (V, E)$
Output: مجموعه مستقل ماکسیمال S_{mis}
1: $S_{mis} \leftarrow \emptyset$
2: $S_{is} \leftarrow \emptyset$ ▷ آغاز مجموعه مستقل
3: $G_s(V_s \leftarrow V, E_s \leftarrow E)$
4: **while** $G_s \neq \emptyset$ **do**
5: $S_{is} \leftarrow$ یک مجموعه مستقل از G_s را انتخاب کن
6: $S_{mis} \leftarrow S_{mis} \cup S_{is}$
7: $V_r \leftarrow S_{is} \cup \{$ همسایه های رئوس در Sis $\}$
8: $E_r \leftarrow \{$ یال های مربوط به رئوس در Vs $\}$
9: $G_s \leftarrow G_s(V_s \leftarrow (V_s - V_r), E_s \leftarrow (E_s - E_r))$

[13]Salihoglu
[14]Garimella
[15]Blelloch

پیاده سازی توزیع شده MIS لوبی درکتابخانهCombBLASبه طور آزادانه در دسترس است. بااین حال،الگوریتمCombBLASبرای فعالیت برروی‌گراف‌های «فیلترشده» طراحی شده است. علاوه براین،چندین پیش‌پردازش از جمله حذفیال‌های خودومتعادل‌سازی بار را نیز انجام می‌دهد. الگوریتم های لوبی توزیع شده که در این مقاله ارائه می‌شوند به طور خاص برای پردازش گرافهای ایستا در مقیاس بزرگ تحت تنظیمات حافظه توزیع شده طراحی شده‌اندو قادر به پردازش گراف‌های بدون ساختار بدون پیش پردازش هستند.

3. الگوریتم‌های لوبی

الگوریتم‌های لوبی از جمله پر استفاده‌ترین الگوریتم‌ها برای یافتن MIS در حافظه مشترک هستند. لوبی در انتشار اولیه الگوریتم خود در مورد یک طرح تکرارشونده کلی و چهار تغییرات خاص بر اساس آن بحث کرد. طرح کلی تکرارشونده در الگوریتم 1 ذکر شده است، در کل این طرح یک مجموعه مستقل غیرتهی را انتخاب می‌کند (خط5) و آن را با خروجی‌ها (S_{mis}) ادغام می‌کند. سپس، مجموعه مستقلی که انتخاب شده است و همسایه‌های آن از گراف ورودی حذف می‌شوند و زیرگراف حاصل از آن برای تکرار بعدی طرح مورد استفاده قرار می‌گیرد (خطوط 7 الی 9). این فرآیند تا زمانی تکرار می‌شود که زیرگراف حاصل خالی باشد. در هر بار تکرار، طرح کلی تکرارشونده یک مجموعه مستقل جدید را تولید می‌کند. لوبی ثابت کرد است که اجتماع تمامی آن مجموعه‌های مستقل یک مجموعه مستقل ماکسیمال است.

برای انتخاب یک مجموعه مستقل از یک زیرگراف در یک تکرار، لوبی دو الگوریتم مونت کارلو را ارائه کرده است که نام آن‌ها: Select A و Select B است. Select B بیشتر برای بهبود ایجاد دو متغیر بیشتر، Select C و Select D مورد استفاده قرار می‌گیرد. تمامی این چهار متغیر از تصادفی‌سازی برای محاسبه یک مجموعه مستقل استفاده می‌کنند. الگوریتم‌های ASelect و BSelect و CSelect غیرقطعی است، در حالیکه Select D قطعی است. در این مقاله، ما بر روی ASelect و BSelect تمرکز کردیم (زیرا C و D متغیرهایی از B هستند). الگوریتم‌های ASelect و BSelect در جدول 1 خلاصه شده‌اند.

ASelect یکی از ساده‌ترین الگوریتم‌ها است. تمامی رئوس (V_s) در زیرگراف را به عنوان مجموعه‌های مستقل در نظر می‌گیرد. سپس، یک عدد تصادفی به شکل $r\ (1 \le r \le |V_s|^4)$ را به هر کدام از رئوس زیرگراف تخصیص می‌دهد. سپس برای هر یال در زیرگراف (یال‌ها در E_s)، ASelect رئوسی در گراف که دارای مقادیر تصادفی بیشتر باشد را حذف می‌کند.

Select B	Select A
1. فرض می‌کند رئوسی که در تست متغیر تصادفی Coin صدق می‌کنند مستقل هستند.	1. فرض می‌کند که تمامی رئوس در یک زیرگراف مستقل هستند
2. مجموعه مستقل را بر اساس درجه توزیع زیرگراف محاسبه می‌کند.	2. مقادیر تصادفی را به رئوس در Vs تخصیص می‌دهد.
	3. مجموعه مستقلی بر اساس مقادیر تصادفی تخصیص یافته را محاسبه می‌کند.

جدول1) معیارهای انتخاب مجموعه مستقل برای الگوریتم Select A و Select B

بر خلاف ASelect، BSelectتمامی رئوس در زیرگراف را به عنوان مجموعه مستقل در یک تکرار در نظر نمی‌گیرد. به جای آن BSelect از یک منغیر تصادفی (Coin[16]) برای تصمیم‌گیری در مورد اینکه آیا رئوس در زیرگراف باید به مجموعه‌ای مستقل انتخاب شوند یا خیر استفاده می‌کند. مقدار Coin بر اساس توزیع احتمالات با استفاده از درجه توزیع زیرگراف تعریف شده است. به طور دقیق‌تر، اگر d(v) درجه تخصیص‌یافته به یک راس $v \in V$`باشد، آنگاه به احتمال 1/2d(v) خواهیم داشت که: Coin(V)=1. اگر d(v) =0 باشد آنگاه Coin(v) همواره 1 خواهد بود. برای جزئیات بیشتر در مورد الگوریتم B، می‌توانید به لوبی اصلی که در [2] منتشر شده بود مراجعه کنید.

A. الگوریتم لوبی در حافظه توزیع‌شده

الگوریتم‌های لوبی مستقیما در اختیار پیاده‌سازی های حافظه توزیع شده موازی قرار نمی‌گیرند. الگوریتم های لوبی باتمرکز بردستگاه های حافظه مشترک و با استفاده ازمدلPRAMمورد تحلیل قرار می‌گیرند. درمدلPRAMتمام

[16] سکه

پردازنده ها پس از خواندن از حافظه مشترک و همچنین قبل از نوشتن درحافظه مشترک همگام سازی می‌شوند. یک روش طبیعی برای گسترش یک الگوریتم توزیع حافظه لوبی به حافظه توزیع شده،استفاده از رویکرد موازی همگام‌سازی توده‌ای[17](BSP) است، در [17] BSPعملیات‌های حافظه مشترک می‌توانند برای مراحل محاسبه،ارتباطات و هماهنگ‌سازی تبدیل شوند. بااین حال،این رویکرد دربسیاری ازمراحل همگام‌سازی مانع ایجاد می‌کند.

یکی دیگر از مشکلات «طرح تکرار شونده کلی» (الگوریتم 1) وابستگی به محاسبات زیرگرافی است. یعنی در هر بار تکرار، الگوریتم یک زیرگراف جدید، با حذف رئوس و یال‌های مجموعه مستقلی که در تکرار فعلی از گراف بودند، ایجاد می‌کند. ساخت یک زیرگراف در هر بار تکرار در حافظه توزیع شده بسیار ناکارآمد است زیرا شامل ارتباطات و همگام‌سازی‌ها است حتی اگر زیرگراف به طور ضمنی از طریق پوشاندن[18] رئوس حفظ و نگهداری شود.

علاوه بر این، الگوریتم Select A نیاز به انتخاب جدیدی از اعداد تصادفی در هر تکرار دارد. محدوده اعداد بستگی به تعداد رئوس باقی‌مانده در زیرگراف دارد. بنابراین، تولید اعداد تصادفی، نیاز به کاهش تعداد رئوس در زیرگراف دارد و این یک مانع در هر تکرار است. علاوه بر این، تولید عدد تصادفی دارای سربار محاسباتی قابل توجهی است.

در بخش بعدی، ما در مورد چگونگی گسترش الگوریتم لوبی به اجرای توزیع‌شده بحث خواهیم کرد و در عین حال از نقایصی که در بالا مطرح شد نیز اجتناب خواهیم کرد. در الگوریتم پیشنهادی، سربار مانع مراحل همگام‌سازی که با همپوشانی محاسباتی و ارتباطی همراه بوده است در اینجا به حداقل رسیده است. محاسبه زیرگراف از طریق فیلترسازی راس بدست خواهد آمد. با این حال فیلترسازی راس، توانایی تکرار در طی ساختار داده گراف به شکل موازی را از کار خواهد انداخت. بنابراین، در پیاده‌سازی ما، ما از یک ساختار داده موازی استفاده کردیم. همچنین نوعی از الگوریتم SelectA را ارائه داده‌ایم که از تولید اعداد تصادفی در هر تکرار جلوگیری می‌کند و از اعداد تصادفی که در آغاز تولید شده بود استفاده می‌کند.

[17]Bulk Synchronous Parallel
[18]masking

4. الگوریتم‌های حافظه توزیع‌شده موازی لوبی

الگوریتم‌های حافظه توزیع‌شده موازی لوبی از یک توزیع 1 بعدی برای توزیع رئوس گراف در بین رتبه‌های شرکت‌کننده‌ها استفاده می‌کند. هر رتبه یک زیرمجموعه از رئوس و یک زیرمجموعه یال مرتبط با رئوس را دریافت می‌کند. در داخل یک رتبه، یک زیرمجموعه راس و یال‌های تخصیص یافته به آن زیرمجموعه با استفاده از یک نمایه گرافی محلی به شکل زیر نمایش داده می‌شود: $(G^{local} = (V^l, E^l))$. یک راس «متعلق به» یک رتبه است و راس‌ها متعلق به رتبه‌های مختلف ارتباطی هستند که به عبور پیام‌ها می‌پردازند. عبور پیام‌های ارتباطی بین رتبه‌ها بر اساس فرآیند BSP طراحی شده است، اما با همپوشانی ارتباطی و محاسباتی برای بهبود کارآیی طراحی شده‌اند.

A. طرح تکرارشونده توزیع کلی

الگوریتم2: طرح تکرارشونده توزیع کلی

```
1: procedure LubyIterate(G^local, select^fn)
2:   buffer ← {}
3:   delete ← {}
4:   while there are NIL vertices in G do
5:     select^fn(&buffer, &delete)
6:     epoch {
7:       for هر راس v در buffer به شکل موازی do
8:         if v is not in delete then
9:           mis[v] ← IN
10:          for برای هر u که در مجاورت قرار دارد (v, G^local) do
11:            Send(u, OUT)
12:    }
13:
14: procedure Receive(v : Vertex, s : State)
15:   mis[v] ← s
```

طرح تکرار شونده توزیع کلی (الگوریتم2) نیاز به محاسبه زیرگراف دارد. اگر چه محاسبه زیر گرافی صریح ممکن است درمحیط های ترتیبی و حافظه مشترک موازی مفید باشد،امادرحافظه توزیع شده به دلیل ایجاد سربار وتوزیع زیرگرافی جدید درهرتکرار،درحافظه توزیع شده ناکارآمداست. قابلیت محاسبه زیرگراف معادل توزیع شده را میتوان با فیلترکردن ریشه‌ها (به عنوان مثال،اعمال یک فیلتر پیش بینی برای نمایش اینکه آیا یک راس درمحاسبات فعلی

درنظر گرفته می‌شود). اگرچه با اعمال فیلتر از بیشتر یال‌ها مجددا باید در هر تکرار عبور کرد، اما افزایش راندمان موازی نسبت به هزینه‌های ناشی از این عبورهای غیرضرروی ارزش دارد.

برای کاهش محاسبات زیرگراف در حافظه توزیع شده،ازدوساختاردادهاستفاده می‌کنیم:

1) یک بافر ضمیمه (append buffer) (Buffer)- برای دسترسی موازی کارآمد و

2)یک ساختار مجموعه (set structure) (حذف مجموعه).

این دو ساختارداده توسط طرح تکراری کلی ایجادشده وبه یکی ازالگوریتمSelect A یا Select B منتقل می‌شود. الگوریتم Select مسئول پرکردن با فرضمیمه است و مجموعه راحذف می‌کند.هنگامی که الگوریتم Selectتصمیم می‌گیردکه یک رأس یک نامزدحضوردرMISباشد،رأس رابه بافر ضمیمه می‌کند. سپس، بعد ازاینکه بافرحاوی مقادیر اولیه رئوس احتمالی MISاست،تمام رأس هایی که یک همسایه بایک مقدار تصادفی کمتراختصاص داده شده دارند،درمجموعه حذف قرارمیگیرند. طرح کلی تکراری به طورموازی بافرضمیمه می‌کند و بررسی می‌کندکه آیا یک رأس درمجموعه حذف شده است یا خیر. اگررأس درمجموعه حذف حضور ندارد،پس رأس بهMISحاصل ضمیمه می‌شود. برای کاهش تردی ددرهنگام استفاده ازمجموعه حذف،مجموعه حذف را به عنوان کلسکسیونی ازمجموعه ها اجر امی‌کنیم،که درآن هر نخ[19]یک مجموعه محلی را به نخ نگه می‌دارد.

افزودن به یک مجموعه حذف، محلی برای فراخوانی نخ است. زمانی که در حال انجام کوئری[20] یک عنصر از یک مجموعه حذف هستیم، ابتدا باید بررسی کنیم که آیا عنصر در مجموعه نخ محلی قرار دارد یا خیر، و سپس باید به دنبال عنصری در مجموعه‌های متعلق به سایر نخ‌ها بپردازیم. در طی مرحله جست‌وجو، مجموعه‌ها تنها خوانده می‌شوند، تا آن‌ها را به شکلی ایمن بتوان بین نخ‌ها به اشتراک گذاشت. طراحی دو مرحله‌ای (بافر و مجموعه حذف) سبب محدودیت مشاجره نخ‌ها بر روی موضوع ورودی سربار کم بر روی بافر ضمیمه می‌شود.

در طی محاسبه،یک رأس می‌تواند دریکی از سه حالت زیر باشد (که دریک نقش هم الکیت ذخیره می‌شود):

1) IN-vertexدرMISاست؛

[19]thread
[20]querying

2) OUT-vertexدرMISنیست؛و

3) NIL-vertexهنوزپردازش نشده است.

درابتداتمامرأس‌هادرحالتNILقراردارند. هنگامی که الگوریتم متوقف می‌شود،تمام حالت‌های رأس بهINیاOUTتغییرمی‌کنند.

اینکه آیا حالت راس باید از NIL به IN یا از NIL به OUT تغییر پیدا کند در داخل طرح تکرارشونده کلی تصمیم‌گیری می‌شود (LubyIterate در الگوریتم2). نخست، طرح‌شونده کلی،الگوریتممناسبSelect، $Select^{fn}$راانتخاب می‌کند ($Select^{A}$ یا $Select^{B}$). الگوریتم انتخابی مشخص مسئول آپلود بافر ضمیمه و مجموعه حذف است (خط 5). طرح کلی تکرارشونده ازطریق بافر ضمیمه به صورت موازی بازنگری می‌کند و بررسی می‌کندکه آیا رأسی درمجموعه حذفوجوددارد (خطوط 7-11). اگررأسی درمجموعه حذف وجود نداشته باشد، حالت راس به IN بروزرسانی می‌شود (خط9). هنگامی که حالت یک راس به حالتINتغییرمی‌کند،تمام حالت‌های همسایه های آن به حالتOUTتغییرمی‌کنند(خط 11). عملیات ارسال تعیین می‌کنند که کدام پیام باید براساس رأس مقصدو توزیع گراف قرارگیرد. پیام‌های فرستاده شده ازطریقSend از طریق Reveive (خط 14-15) دریافت می‌شوند. پیمایش بین رأس‌ها دربافرضمیمه صورت می‌گیردوبه‌روزرسانی حالت های رأس دریک مرحله فوقالعاده تک (به عنوان مثال،دری کدوره تک) اتفاق می‌افتد.

B. Select A توزیع‌شده

الگوریتم Select A(Luby(A)) یک نمایه از گراف‌محلی، یک بافر ضمیمه و یک مجموعه حذف را می‌گیرد. الگوریتم SelectA در الگوریتم 3 بیان شده است. G^{Local}نمایه گراف محلی است، abufferنشان‌دهنده بافر ضمیمه است و deletsetنشان‌دهنده مجموعه حذف است. الگوریتم Select A نخست تعداد رئوسی که در حالت NIL قرار دارند را با استفاده از یک Global Reduction محاسبه می‌کند (خطوط 4 الی 7). سپس، برای هر راسی که در حالت NIL قرار دارد، یک مقدار تصادفی به شکل K ($1< k < globalcount^4$) تخصیص داده

می‌شود (خطوط 10 الی 13). زمانی که مقادیر تصادفی تولید شد، ترکیبی از Id رده‌ها و Id نخ‌ها به منظور تولید یک مقدار تصادفی منحصربه‌فرد دانه[21] برای هر نخ مورد استفاده قرار می‌گیرد (()Seedدر خط 8). مقادیر تصادفی در یک نقشه مالکیت ذخیره‌سازی می‌شوند. یک رده تنها مقادیر تصادفی برای رئوس را در گراف محلی ذخیره‌سازی می‌کند. درحین تخصیص یک مقدار تصادفی به هر راس محلی در حالت NIL، الگوریتم SelectA این رئوس را به بافر ضمیمه وارد می‌کند (خط13).

الگوریتم3: ^{A} Select توزیع‌شده

```
1: procedure Select^A(G^local, ref abuffer, ref deleteset)
2:   localcount ← 0
3:   globalcount ← 0
4:   for each Vertex v in G^local in parallel do
5:     if mis[v] == NIL then
6:       localcount ← (localcount + 1)
7:   globalcount ← Reduce(localcount,SUM)
8:   random ← RandomDist(1, globalcount^4. Seed())
9: /*          NIL     تخصیص مقادیر تصادفی به رئوس در حالت                  */
10:  for each Vertex v in G^local in parallel do
11:    if mis[v] == NIL then
12:      π[v] ← random.Value()
13:      abuffer.add(v)
14: /*  استفاده از مقادیر تصادفی برای حذف مغایرت در رئوس
          */
15:  epoch {
16:    for each Vertex i in abuffer in parallel do
17:      for each j in adjacencies(v, G^local) do
18:        if u belongs to G^local then
19:          if π[i] >= π[j] then
20:            deleteset.add(i)
21:          else
22:            deleteset.add(j)
23:        else
24:          Send(j, i, π[i], COMPARE)
25:  }
26: /*  هر فراخوانی ارسال، یک دریافت را فراخوانی می کند          */
27: procedure Receive(j:Vertex, i:Vertex, irnd:Real, act:Action)
28:   if act == COMPARE then
29:     if irnd >= π[j] then
30:       Send(i, j, π[j], REMOVE)
31:     else
32:       deleteset.add(j)
33:   else
34:     if act == REMOVE then
35:       deleteset.add(j)
```

[21] seed

درمرحله بعد،الگوریتم به صورت موازی ازطریق رأس‌هایی ازبافر ضمیمه عبورمی‌کند ومجاورهایی رابا مقادیر تصادفی بالاترکه درحالتNILدرمجموعه حذف شده (خطوط 16-24) هستند قرارمی‌دهد. اگررأس مجاور بهG^{local}تعلق ندشته باشد،یک پیامی به موقعیت مناسب (خط 24) ارسال می‌شود

که شامل راس منبع i id و مقدار تصادفی آن است. رده‌ای که متعلق به راس مقصد j است بررسی می‌کند که آیا مقادیر تصادفی دریافت شده کمتر از مقدار تصادفی j است یا خیر. اگر اینگونه باشد، j به مجموعه حذف اضافه می‌شود، در غیر این صورت پیامی به i برگشت داده می‌شود تا i را به مجموعه حذف اضافه کند. پیام اول نشان‌دهنده استفاده از عمل COMPARE[22] است و عمل دوم با استفاده از عمل REMOVE[23] صورت می‌پذیرد. هر تماس ارسالی به یک تابع Receive[۲۴] متناظر مرتبط می‌شود (خطوط 27 الی 35). در پایان اجرای هر دوره در خطوط 15 الی 25، رئوسی که در بافر ضمیمه هستند و نه رئوسی که در بافر حذف هستند، یک مجموعه مستقل هستند.

C. انتخاب AV (یک دگرگونی از Select A).

الگوریتم Select Aمقادیری تصادفی در هر تکرار تولید می‌کند. تولید عدد تصادفی از نظر محاسباتی گران است، همچنین برای تولید اعداد تصادفی نیاز به محاسبه حجم کلی مجموعه زیرگراف داریم (در طی تمامی رده‌های توزیع شده) که سربار ارتباطات اضافی آن به علت کاهش توزیع و مانع‌هایی در همگام‌سازی رده‌ها است.

Select AV(lubbu (AV)) تقریبا همانند Select A است با این تفاوت که ما اعداد تصادفی برای محاسبه یک مجموعه مستقل را تولید نمی‌کنیم. به جال آن، ما از IDهای رئوس نمایش‌دهنده گراف به منظور شکستن تقارن و محاسبه یک مجموعه مستقل استفاده خواهیم کرد. به عبارت دیگر، $\pi[i] = i \; \forall i \in V_s$ که در آن V_Sنشان‌دهنده مجموعه رئوس یک زیرگراف است. این روش بستگی به توزیع شناسایی‌کننده راس در ساختار داده گراف دارد اما به خوبی با نمایش ما نیز کار می‌کند که به شکل تصادفی به جایگشت رئوس پیش از شروع الگوریتم می‌پرداخت.

---

[22] مقایسه
[23] حذف
[24] دریافت

D. SelectB توزیع‌شده

الگوریتم 4: $Select^B$توزیع‌شده

```
1: procedure Select^B(G^local, ref abuffer, ref deleteset)
2:   degree ← {} /*  محاسبه درجه راس که مرتبط با زیر گراف است  */
3:   epoch {
4:     for each Vertex v in G^local in parallel do
5:       if mis[v] == NIL then
6:         for each u in adjacencies(v, G^local) do
7:           if u belongs to G^local then
8:             if mis[u] == NIL then
9:               degree[v] ← (degree[v] + 1)
10:            else
11:              Send^1(u, v, ISNIL)
12:  }
13: /*  ایجاد مجموعه ای مستقل بر اساس مقدار سکه  */
14:  for each Vertex v in G^local in parallel do
15:    if coin(v, degree[v]) == 1 then
16:      abuffer.add(v)
     /*  حذف رئوسی که دارای مغایرت هستند  */
17:  epoch {
18:    for each Vertex i in abuffer in parallel do
19:      for each j in adjacencies(v, G^local) do
20:        if j belongs to G^local then
21:          if j is in abuffer then
22:            if degree[i] <= degree[j] then
23:              deleteset.add(i)
24:            else
25:              deleteset.add(j)
26:        else
27:          Send(j, i, degree[i], COMPARE)
28:  }
29: /*  هر تماس Send1 یک Receive1 را فراخوانی می کند  */
30: procedure Receive^1(u:Vertex, v:Vertex, act:Action)
31:   if act == ISNIL then
32:     if mis[u] == NIL then
33:       Send^1(v, u, NILTRUE)
34:   else
35:     if act == NILTRUE then
36:       degree[u] ← (degree[u] + 1)
37: /*  هر تماس Send یک Receive را فراخوانی می کند  */
38: procedure Receive(j:Vertex, i:Vertex, irnd:Real, act:Action)
    ▷ /*  همانند روند دریافت در الگوریتم 3 است  */.
```

الگوریتم Select B (یا Luby (B)) تمامی رئوسی که در حالت NIL هستند را به بافر ضمیمه، اضافه نمی‌کند و به جای آن تنها یک زیرمجموعه‌ای از رئوسی که در حالت NIL هستند را اضافه می‌کند. زیرمجموعه بر اساس مقدار تصادفی Coin محاسبه می‌شود. مقدار تصادفی Coin دارای دو مقدار 1 و 0 است که بر اساس درجه توزیع راس‌ها

تخصیص داده می‌شود. بنابراین، الگوریتم Select B نخست درجه هر راس مرتبط با زیرگراف را محاسبه می‌کند. سپس الگوریتم زیرمجموعه‌ای از رئوس را در حالت NIL انتخاب می‌کند تا یک مجموعه مستقل بر اساس مقدار Coin داشته باشد. پس از آن، الگوریتم به حالت موازی بررسی می‌کند تا ببیند که آیا رئوسی مجاور انتخاب شده است یا خیر. اگر اینگونه باشد، الگوریتم از درجه رئوس برای حل مغایرت استفاده می‌کند و تمامی رئوسی که غیر مستقل هستند را حف می‌کند. شبه کد الگوریتم برایالگوریتم Select B در الگوریتم 4 ارائه شده است.

محاسبهدرجهرأسدرزیرگرافجاریازبهبرقراریارتباطبارده‌هایدورادارد. خطوط 4-11 محاسباتدرجهرانشانمی‌دهد. اگریکرأسمجاوردرمحلفعلینیست،الگوریتمیکپیامرابهمکاندورافتادهارسالمی‌کندتا راس مجاوربررسیشود (خط 11) و این کار با استفاده از عمل ISNIL برای انجام کوئری در مورد وضعیت راس مجاور صورت می‌گیرد. این پیام‌ها روند Receive 1 را در مکان دور افتاده احضار می‌کنند (خطوط 30 الی 36). اگر راس مجاور در حالت NIL باشد، رتبه دورافتاده دستوری با پاسخ NILTRUE را بر می‌گرداند.

خطوط 14-16 نشان می‌دهند که چگونه الگوریتمSelect B، یک Coin را احضار می‌کندکه یک متغیرتصادفی است تا زیرمجموعه ای از رأس‌ها رابه عنوان نامزد برای یک مجموعه مستقل انتخاب کند. تابعCoinیک رأس و در جهآن رابرای تصمیم گیری درموردااینکه آیامقدارمتغیرتصادفی 1 یا 0 است،دریافت می‌کند. اگرتابع1Coin رابرگرداند،رأس به بافر ضمیمه اضافه می‌شود.

پس از افزودن زیرمجموعه‌ای از راس‌ها در زیرگراف بافر ضمیمه، الگوریتم Select B از طریق محتوای موجود در بافر ضمیمه به شکل موازی تکرار می‌شود. اگر راسی در بافرض میمه دارای راسی مجاور باشد که در بافر ضمیمه نیز ظاهر شده باشد، در آن هنگام راسی با درجه کوچکتر حذف خواهد شد. راسی که دارای درجه بالاتر باشد به مجموعه حذف (خطوط 18 الی 27) اضافه خواهد شد. اگر راس مجاور در یک مکان دور افتاده باشد یک پیام به آن ارسال می‌شود (خط 27). کد دریافتی برای پردازش پیام‌های ارسال شده (خط 27) مشابه با روند Receive در Select A است (خطوط 27 الی 35 در الگوریتم A). مانند Select A، زمانی که Select B اجرای خودش را در هر دوره

از خطوط 17 الی 28 به پایان می‌رساند، راس‌هایی که در بافر ضمیمه هستند اما در مجموعه حذف نیستند، نشان‌دهنده یک مجموعه مستقل هستند.

5. آزمایش‌ها و نتایج

A. پیاده‌سازی

الگوریتم‌های پیشنهادی بر روی یک چهارچوب پیام‌نگاری فعال با نام AM++[3] با استفاده از pthreads برای نخ‌سازی در داخل گره اجرا شده است.

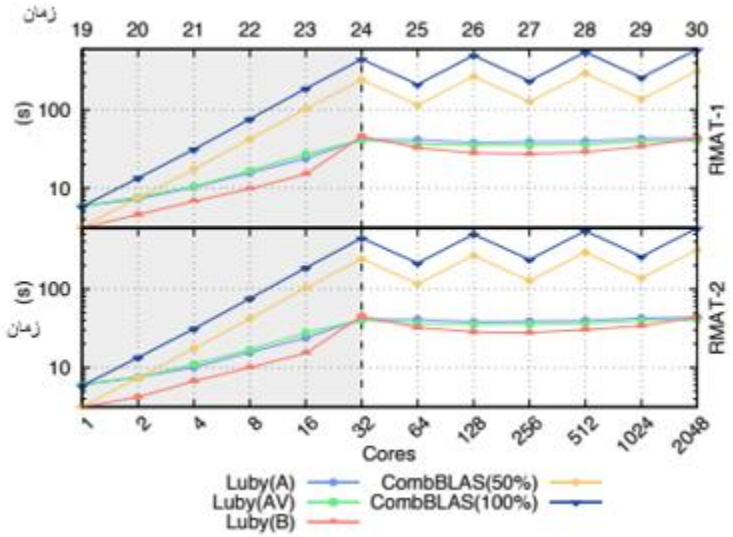
رئوس گراف به طور مساوی بین گره‌های شرکت‌کننده (توزیع 1 بعدی بلاک) توزیع شده‌اند. گراف محلی با استفاده از قالب ردیف اسپارس[۲۵] فشرده (CSR) نمایش داده شده است. هر یال بی‌جهت با استفاده از یال‌های دو طرفه نشان داده است.

در پیاده سازی الگوریتم نیازی به پیش‌پردازش ورودی ها نیست و میتواند بایال‌های موازی و حلقه‌هایی به خود (مفاهیم عمومی درورودی های مصنوعی) مقابله کند. هرگاه کدی وجودداشته باشدکه ازطریق مجاورت‌های یک راس تکرار شود،الگوریتم راس‌های مجاور را به یک مجموعه محلی اضافه می‌کند. بدنه حلقه فقط درصورتی اجرا می‌شودکه رأس مجاور درمجموعه وجودنداشته باشد (کد زیر را ببینید).

```
1: ...
2: adjacentvertices ← {}
3: for each u in adjacencies(v, G^local) do
4:   if (u! = v) then                          ▷ /*Exclude self-loops*/
5:     if u not in adjacentvertices then▷ /*Exclude parallel edges*/
6:       adjacentvertices.insert(u)
7:       ...
```

```
1: ...
2: adjacentvertices ← {}
3: for each u in adjacencies(v, G^local) do
4:   if (u! = v) then                          ▷ /*  مانع از حلقه های به خود  */
5:     if u not in adjacentvertices then▷ /*  مانع از یال های موازی  */
6:       adjacentvertices.insert(u)
7:       ...
```

[25] sparse



شکل2) نتایج مقایس‌بندی ضعیف الگوریتم‌های MIS برای گراف‌های RMAT، از جمله FilteredMIS. مناطق سایه‌دار نشان‌دهنده اجرای حافظه مشترک است.

B. تنظیمات آزمایش

ما آزمایش را بر روی یک سیستمCray XCبا 2 پردازندهBroadwell22 هسته ای این تلXeonو 128 گیگابایتDDR4-2400 حافظه درهر گره انجام دادیم. برای مقیاس بندی نتایج،ما تنها تا 32 هسته درهر گره برای ارائه پیمایش یکنواخت استفاده کردیم. ماازCray MPICH MPIنسخه 4 .4 .7 و 0 .3 .6 GCCاستفاده کردیم. ما ازدوپردازش درهرگره استفاده کردیم(یکی به ازای هر حوزه NUMA) و از MPI در حالت چند نخی استفاده کردیم.

C ورودی گراف

ماالگوریتم‌هایMISرادرشرایط پیمایش ضعیف و قوی ارزیابی می‌کنیم. ما ازگراف‌های مصنوعیRMAT[18]استفاده می‌کنیم. دونوع گرافی مصنوعیRMATاستفاده شده است. آن‌ها عبارتند از:

• RMAT-1: گراف‌هایی که بر اساس Graph 500 [19] فعلی هستند معیار جست‌وجوی اول سطح با پارامترهایی به شکل A= 0 .57 و B=C= 0 .19 و D= 0 .05 دارند.

• RMAT-2: گراف‌های تولیدشده بر اساس معیارهای SSSPGraph[20] با پارامترهای RMAT به شکل زیر هستند: A= 0. 50 و B=C= 0. 1 و D= 0. 3 است.

هر دو نوع از گراف‌ها دارای 16 یال بی‌جهت به ازای هر راس هستند. نتایج پیمایش قوی با RMAT-1 و RMAT-2 مقیاس 25 گرافی اجرا شده است (که بزرگترین مقدار ممکن پیش از خستگی حافظه است).

D نتایج پیمایش ضعیف

برای پیمایش ضعیف ما پیاده‌سازی خودمان را با پیاده‌سازی FilteredMIS[14] در کتابخانه CombBLAS[4] مقایسه کردیم. الگوریتم FilteredMIS بر روی Luby (A) با فیلترسازی یال‌ها انجام شد. با این حال پیاده‌سازی که در این مقاله ارائه شده است هیچ‌گونه فیلتری را بر یال‌ها انجام نمی‌دهد. ما نتایج FilteredMIS را با 0 ٪ و 50٪ فیلتر یال‌ها را نشان دادیم در حالیکه هیچ کدام از یال‌ها و نصف آن‌ها نادیده گرفته شده بود. هر چقدر بیشتر یال‌ها نادیده گرفته شوند، FilteredMIS عملکردی بهتر دارد.

شکل 2 نشان‌دهنده نتایج پیمایش ضعیف از Luby (A)، Luby (B) و Luby (C) و FilteredMIS برای ورودی‌های گراف RMAT-1 و RMAT-2 است. برای هر دو ورودی گراف توزیع‌شده الگوریتم‌های لوبی در این مقاله بهتر از FilteredMIS

و CombBLAS عمل کردند (هم برای میزان فیلترسازی 50٪ و هم برای میزان فیلترسازی 100٪).

نتایج اجرای توزیع‌شده FilteredMIS نشان‌دهنده یک الگوی زیگ‌زاگ[26] است (زمانی که هسته‌ها بیشتر از 32 هستند). در نسخه CombBLAS ما فقط از یک تعداد مربع از وظایف پشتیبانی می‌کنیم، بنابراین، اجرای آن بر روی تعداد اعداد غیر مربعی گره‌ها (2 یا 8 یا 32) ما باید از دو وظیفه به ازای هر گره استفاده کنیم تا به تعداد مربع

[26] zig-zag

از فرآیندها دست پیدا کنیم. هنگامی که تعداد وظایف در هر گره 2 است، زمان اجرای FilteredMIS کاهش پیدا می‌کند و زمانی که تعداد وظایف در هر گره 1 است، زمان اجرا افزایش پیدا می‌کند. ما قابلیت چندنخی [۲۷] در CombBLAS را فعال کردیم تا وظایف بتوانند از مزایای هسته‌های چندگانه استفاده کنند. ما متوجه شدیم که CombBLAS بدترین عملکرد را با یک وظیفه به ازای هر هسته دارد (زمانی که قابلیتچندنخی وجود نداشته باشد).

همانطور که از شکل 2 می‌توان متوجه شد، Luby (B) در حافظه اشتراکی برای هر دو گراف ورودی بهتر عمل می‌کند (زمانی که تعداد هسته‌ها کمتر از 32 باشد). بر خلاف Luby (A)، Luby (B) تمامی رئوس را در زیرگراف به عنوان تقریب اولیه‌ای از مجموعه‌ای مستقل در نظر نمی‌گیرد. Luby (B) دارای یک گام انتخابی است، که زمانی است که یک زیرمجموعه از رئوس زیرگراف را بر اساس توزیع احتمالات محاسبه می‌کند (بخش 3، تابع «Coin» را ببینید). از آنجایی که Luby (B) یک زیرمجموعه از رئوس زیرگراف را انتخاب می‌کند، Luby (B) قادر به محاسبه یک مجموعه مستقل سریع‌تر از Luby(A) در تکرار است. از سویی دیگر، گام انتخابی ندارد و تمامی رئوس را در زیرگراف به عنوان نامزدی برای مجموعه مستقل در نظر می‌گیرد. آمارهای داده‌ای که ما جمع کردیم نشان می‌ده که Luby (A) تمامی زمان را صرف محاسبه مجموعه‌های مستقل در تکرار اول می‌ند، به خصوص زمانی که در حافظه مشترک اجرا می‌شود. برای مثال، آمار در جدول 2 برای گراف RMAT-1 20 در دو هسته جمع‌آوری شده است. سایر مقیاس‌ها نیز رفتاری مشابه دارند.

Luby(B)	Luby (A)	
4. 17	7. 13	زمان اجرا (ثانیه)
0 . 01	6	زمان برای تکرار 0 ام (ثانیه)
517394	1048576	رئوسی که در تکرار 0 ام هستند
2043	917623	اندازه مجموعه حذف شده

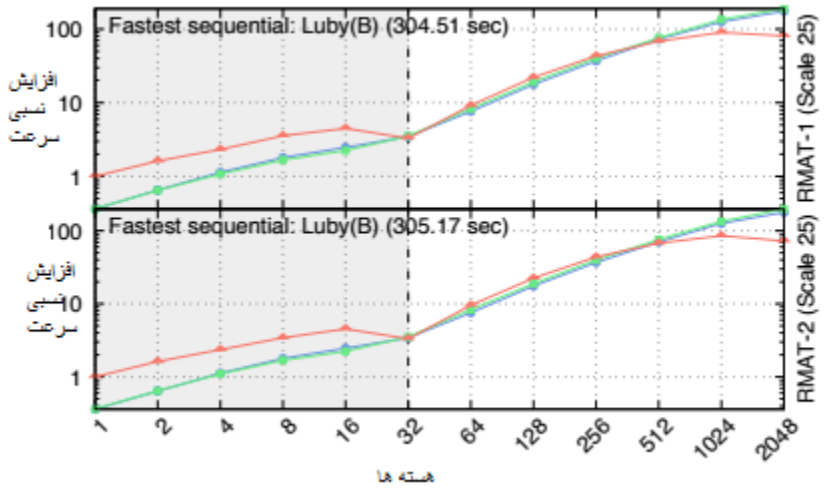
جدول2) آمارهای مرتبط با زمان‌اجرا برای Luby (A) و Luby (B) بر روی RMAT-1، گراف 20 مقیاسی در 2 هسته

[27] multi-threading

Luby (A) با این حال خیلی سریع‌تر از Luby (B) همگرا می‌شود. همانطور که در جدول 2 نشان داده شده است،

Luby (A) 6 تکرار نیاز دارد تا به پایان برسد در حالیکه Luby (A)به 20 تکرار نیاز دارد. زمانی که تعداد تکرارها بالا باشد، سربار همگام‌سازی کلی نیز افزایش می‌یابد. در مقیاس 24 (زمانی که هسته‌ها 32تا باشند) ما شاهد افزایش ناگهانی زمان اجرای Luby (A) خواهیم بود. این بدین دلیل است که مقایس 24 در 2 فرآیند اجرا می‌شود و سربار اطلاعات قابل توجه است. عملکرد Luby (B) در مقایس 24 بیشتر از عملکرد Luby (A) تحت تاثیر قرار می‌گیرد، زیرا تعداد تکرارها در Luby (B) بیشتر است بنابراین سربار همگام‌سازی نیز در Luby (B) مهم‌تر از Luby (A) است.

تفاوت بین Luby (A) و Luby (B) برجسته نیست. Luby (AV) قادر به دستیابی به بهبودی جزئی نسبت به Luby (A) در اجرای توزیع‌شده است. دلیل اصلی آن هم این است که Luby (AV) نیاز به محاسبه اندازه مجموعه رئوس را با استفاده از یک Glibal Reduction ندارد و همچنین نیاز به تولید اعداد تصادفی نیز ندارد.



شکل3) پیمایش قوی ناشی از الگوریتم‌های MIS برای RMAT-1 و RMAT-2، ورودی‌های گراف 25 مقایسی، نواحی سایه‌خورده نشان‌دهنده اجرای حافظه اشتراکی است.

تمامی الگوریتم‌های Luby (A, AV , B) در این مقاله عملکردی مناسب برای پیمایش ضعیف در حافظه اشتراکی را نشان نمی‌دهند زیرا مغایرت ایجاد شده در ساختار داده سبب افزایش تعداد نخ‌ها شده است. با این حال ما برای تمامی الگوریتم‌های Luby (A, AV , B) در مورد پیمایش ضعیف در حافظه توزیع‌شده عملکردی مناسب را شاهد هستیم.

E. نتایج پیمایش قوی

برایآزمایش‌های پیمایشقوی،ماالگوریتمهایMISراروی‌مقادیر RMAT-1 و RMAT-2 گراف‌های 25 مقیاسی اجرا کردیم. برایدرکبهترنحوه مقیاس‌بندی نسبی الگوریتم‌ها نسبت به یکدیگر، ما افزایش سرعت نسب را اندازه‌گیری کردیم که برای مثال برابر با $\frac{T_{ref\_1}}{T_n}$ است. نسبت زمان اجرای الگوریتمی با سریع‌ترین دنباله به شکل T_{ref_1}است و زمان اجرای موازی در پردازش n عنصر به شکل T_nاست.

شکل3 نشان‌دهنده نتایج پیمایش قوی الگوریتم‌های MIS است که در این مقاله برای ورودی‌های گرافی که در بالا مورد بحث قرار گرفت ارائه شده است. به دلیل سربار همگام‌سازی که در بالا در مورد پیمایش ضعیف مورد بحث قرار گرفت، luby(B) افزایش سرعت بهتری در حافظه اشتراکی دارد، اما در حافظه توزیع‌شده افزایش سرعت آن در پیمایش‌های بالاتر به دلیل مشکل میزان سربار بالا در همگام‌سازی کاهش می‌یابد.

6. نتیجه‌گیری

بسیاری از تحقیقات فعلی در حوزه MIS بر روی تحلیل تئوری به جای پیاده‌سازی عملی تمرکز داشتند. علاوه بر این، تنها موارد محدودی که در مورد پیاده‌سازی عملی بودند، اکثرا الگوریتم MIS لوبی را پیاده‌سازی کرده بودند. MIS لوبی به دلیل سربار همگام‌سازی و سربار محاسبات زیرگرافی، به شکل موثر و بلافاصله‌ای در الگوریتم‌های موازی حافظه توزیع‌شده گسترش نمی‌یافت.

در این مقاله ما نسخه‌های توزیع‌یافته الگوریتم‌های لوبی موازی را ارائه دادیم. الگوریتم‌ها به منظور به حداقل‌رسانی سربار همگام‌سازی با همپوشانی ارتباطی و محاسباتی و با به حداقل‌رسانی سربار محاسباتی زیرگراف‌ها با استفاده از فیلترسازی رئوس و با حفظ ساختارهای داده‌های موازی فعالیت می‌کنند. نتایج ما نشان‌دهنده این است که الگوریتم‌هایی که ما ارائه دادیم، چندین برابر سریع‌تر از الگوریتم‌های فعلی MIS هستند.

REFERENCES

[1] S. A. Cook, "A taxonomy of problems with fast parallel algorithms," *Information and control*, vol. 64, no. 1, pp. 2–22, 1985.

[2] M. Luby, "A simple parallel algorithm for the maximal independent set problem," *SIAM journal on computing*, vol. 15, no. 4, pp. 1036–1053, 1986.

[3] J. J. Willcock, T. Hoefler, N. G. Edmonds, and A. Lumsdaine, "AM++: A Generalized Active Message Framework," in *Proc. 19th Internat. Conference on Parallel Architectures and Compilation Techniques*. ACM, 2010, pp. 401–410.

[4] A. Buluç and J. R. Gilbert, "The combinatorial blas: Design, implementation, and applications," *International Journal of High Performance Computing Applications*, p. 1094342011403516, 2011.

[5] M. Goldberg and T. Spencer, "Constructing a maximal independent set in parallel," *SIAM Journal on Discrete Mathematics*, vol. 2, no. 3, pp. 322–328, 1989.

[6] A. Goldberg, S. Plotkin, and G. Shannon, "Parallel symmetry-breaking in sparse graphs," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 1987, pp. 315–324.

[7] M. K. Goldberg, "Parallel algorithms for three graph problems," *Congressus Numerantium*, vol. 54, no. 111-121, pp. 4–1, 1986.

[8] N. Alon, L. Babai, and A. Itai, "A fast and simple randomized parallel algorithm for the maximal independent set problem," *Journal of algorithms*, vol. 7, no. 4, pp. 567–583, 1986.

[9] R. M. Karp and A. Wigderson, "A fast parallel algorithm for the maximal independent set problem," *Journal of the ACM (JACM)*, vol. 32, no. 4, pp. 762–773, 1985.

[10] G. E. Blelloch, J. T. Fineman, and J. Shun, "Greedy Sequential Maximal Independent Set and Matching Are Parallel on Average," in *Proceedings of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*. ACM, 2012, pp. 308–317.

[11] N. A. Lynch, *Distributed algorithms*. Morgan Kaufmann, 1996.

[12] Y. Métivier, J. M. Robson, N. Saheb-Djahromi, and A. Zemmari, "An optimal bit complexity randomized distributed mis algorithm," *Distributed Computing*, vol. 23, no. 5-6, pp. 331–340, 2011.

[13] F. Kuhn, T. Moscibroda, T. Nieberg, and R. Wattenhofer, "Fast deterministic distributed maximal independent set computation on growth-bounded graphs," in *International Symposium on Distributed Computing*. Springer, 2005, pp. 273–287.

[14] A. Buluc, E. Duriakova, A. Fox, J. R. Gilbert, S. Kamil, A. Lugowski, L. Oliker, and S. Williams, "High-productivity and high-performance analysis of filtered semantic graphs," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 237–248.

[15] S. Salihoglu and J. Widom, "Optimizing graph algorithms on pregel-like systems," *Proceedings of the VLDB Endowment*, vol. 7, no. 7, pp. 577–588, 2014.

[16] K. Garimella, G. De Francisci Morales, A. Gionis, and M. Sozio, "Scalable facility location for massive graphs on pregel-like systems," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 273–282.

[17] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[18] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining." in *SDM*, vol. 4. SIAM, 2004, pp. 442–446.

[19] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the graph 500 benchmark," *Cray User's Group (CUG)*, 2010.

[20] Graph500Contributors. (2016) Graph 500 benchmark 1 ("search"). [Online]. Available: http://www.cc.gatech.edu/ jriedy/tmp/graph500/