بسم الله الرحمن الرحیم

www.jozve.org

www.jozve.org

دانشگاه یزد

دانشکده‌ی ریاضی

گروه علوم کامپیوتر

پایان‌نامه

جهت دریافت درجه کارشناسی ارشد

علوم کامپیوتر

بررسی الگوریتم‌های هندسی موازی برای کامپیوترهای چندهسته‌ای

استاد راهنما:

دکتر محمد فرشی

استادان مشاور:

دکتر سید ابوالفضل شاهزاده‌فاضلی

دکتر فضل ا... ادیب نیا

پژوهش‌گر:

فاطمه دهقانی فیروزآبادی

اسفند ۱۳۹۰

www.jozve.org

www.jozve.org

تقدیم به پدرم

که در پناهش، نه از سرمای زمستان، بر من گزندی رسید؛ و نه از گرمای آتشین تابستان...

تقدیم به مادرم

به پاس تمام نیایش‌های شبانه‌اش، مهربانی دریغش و عشق بی‌نهایتش

تقدیم به همسر عزیزم

برای تمام صبر، اندیشه و ایمانی که نثارم کرد...

www.jozve.org

www.jozve.org

سپاس‌گزاری

سپاس خداوند را که مرا به راه دانش رهنمون شد و هموست که فرصت شاگردی اساتید گرامی را بر من ارزانی داشت تا چراغی بر تاریکی جهلم باشند.

در آغاز وظیفه خود می‌دانم از زحمات بی‌دریغ استاد راهنمای خود، جناب آقای دکتر محمد فرشی، صمیمانه تشکر و قدردانی کنم که قطعاً بدون راهنمایی‌های ارزنده ایشان، این مجموعه به انجام نمی‌رسید. سعه‌ی صدر، اخلاق و نکته‌سنجی این بزرگوار را سرلوحه خود در تمام مراحل زندگی قرار داده و از خدای منان برای ایشان، توفیق و سربلندی روزافزون را خواستارم.

از همراهی و راهنمایی‌های استادان فرهیخته، جناب آقای دکتر سید ابوالفضل شاهزاده‌فاضلی و جناب آقای دکتر فضل ا... ادیب‌نیا که زحمت مطالعه و مشاوره این پایان‌نامه را تقبل فرمودند و در آماده‌سازی این رساله، به نحو احسن اینجانب را مورد راهنمایی قرار دادند، کمال سپاس و امتنان را دارم.

www.jozve.org

www.jozve.org

چکیده

غشای محدب یک مجموعه‌ای از نقاط، کوچکترین مجموعه محدبی است که همه‌ی نقاط را شامل می‌شود. غشای محدب یک ساختار اولیه در ریاضیات و هندسه محاسباتی است و در مسائلی مانند تشخیص الگو، شکل‌شناسی و پردازش تصویر کاربرد فراوانی دارد.

در این پایان‌نامه، یک الگوریتم موازی مقیاس‌پذیر برای ساخت غشای محدب مجموعه‌ای از n نقطه در صفحه بررسی می‌گردد. این الگوریتم برای مدل چندکامپیوتری دانه‌درشت طراحی شده است که در این مدل، p پردازنده هر یک با حافظه محلی $O\left(\frac{n}{p}\right) \gg O\left(1\right)$ ، توسط شبکه‌های ارتباطی دلخواه به یکدیگر متصل می‌شوند. این الگوریتم برای دامنه وسیعی از مقادیر n و p مقیاس‌پذیر است یعنی برای مقادیر $\frac{n}{p} \geq p^{\epsilon}$، ($\epsilon > 0$) کارا و قابل اجراست. زمان مورد اجرای این الگوریتم برابر $O\left(\frac{T_{sequential}}{p} + T_s\left(n,p\right)\right)$ است که $T_{sequential}$ زمان اجرای بهترین الگوریتم ترتیبی و$T_s\left(n,p\right)$ زمان مرتب‌سازی کلی n داده بر روی یک ماشین p پردازنده‌ای است. علاوه بر این، الگوریتم تنها از تعداد ثابتی دورهای ارتباطی کلی استفاده می‌کند.

www.jozve.org

www.jozve.org

فهرست مطالب

۱ مقدمه‌ای بر برنامه‌نویسی موازی **۱**

۱.۱ محاسبه موازی ۲

۲.۱ معماری کامپیوتر موازی ۵

۱.۲.۱ سیستم‌های حافظه مشترک ۷

۲.۲.۱ سیستم‌های حافظه توزیع شده ۸

۳.۱ شبکه‌های ارتباطی داخلی ۸

۴.۱ مدل‌های برنامه‌نویسی موازی ۹

۱.۴.۱ مدل حافظه مشترک ۱۰

۲.۴.۱ مدل انتقال پیام ۱۲

۵.۱ الگوریتم‌های موازی ۱۳

۶.۱ معیار ارزیابی سیستم موازی ۱۴

۷.۱ دو نمونه از الگوریتم موازی ۱۵

۱.۷.۱ الگوریتم محاسبه حاصل جمع آرایه‌ای از اعداد در مدل $EREWPRAM$. ۱۶

۲.۷.۱ الگوریتم محاسبه حاصل جمع آرایه‌ای از اعداد در مدل انتقال پیام ۱۷

۸.۱ تکنولوژی چندهسته‌ای ۱۹

۱.۸.۱ تکامل تکنولوژی چندهسته‌ای ۲۰

۲.۸.۱ برنامه‌نویسی سیستم‌های چندهسته‌ای ۲۲

۲ معرفی مدل برنامه‌نویسی مورد نیاز **۲۵**

www.jozve.org

۱.۲ مفهوم دانه‌درشت و دانه‌ریز ۲۶
۲.۲ مدل‌های محاسباتی دانه‌ریز ۲۷
۳.۲ مدل‌های محاسباتی دانه‌درشت ۳۰
۱.۳.۲ مدل BSP ۳۰
۲.۳.۲ مدل CGM ۳۲
۴.۲ طرح کلی یک الگوریتم در مدل CGM ۳۳
۵.۲ انواع عملیات ارتباطی در مدل CGM ۳۵
۱.۵.۲ مفاهیم کلی ارتباطات ۳۵
۲.۵.۲ عمل ارتباطی اصلی: مرتب‌سازی کلی ۳۷
۳.۵.۲ سایر عملیات ارتباطی برپایه مرتب‌سازی کلی ۳۹

۳ الگوریتم موازی محاسبه غشای محدب ۵۱

۱.۳ محاسبه غشای محدب ۵۲
۱.۱.۳ غشای محدب یک مجموعه از نقاط در صفحه ۵۳
۲.۱.۳ الگوریتم ترتیبی محاسبه غشای محدب ۵۵
۲.۳ طرح کلی الگوریتم موازی ۵۶
۳.۳ ترکیب موازی غشاهای محدب ۵۷
۱.۳.۳ تعاریف و مفاهیم مورد نیاز الگوریتم ترکیب ۵۸
۲.۳.۳ رویه محاسبه عنصر بعدی ۶۱
۳.۳.۳ توصیف جدیدی از غشای فوقانی ۶۳
۴.۳.۳ مجموعه جداکننده‌ها و معرفی g_i^* ۶۶
۵.۳.۳ رویه محاسبه g_i^* ۶۹
۶.۳.۳ الگوریتم $MergeHull1$ ۸۰
۷.۳.۳ الگوریتم $MergeHull2$ ۸۴

۴ بررسی کارایی عملی الگوریتم‌ها ۹۱

www.jozve.org

۵ نتایج و پیشنهادات ۹۷

واژه‌نامه فارسی به انگلیسی ۱۰۱

واژه‌نامه انگلیسی به فارسی ۱۰۴

مراجع ۱۰۷

www.jozve.org

www.jozve.org

لیست تصاویر

۱.۱ محاسبات سریال ۲
۲.۱ محاسبات موازی و کاربرد آن ۳
۳.۱ تصاویری از داده‌های محاسباتی در رشته‌های مختلف ۳
۴.۱ تصاویر ایجاد شده توسط نرم‌افزارهای تجاری ۴
۵.۱ انواع معماری‌های MIMD ۷
۶.۱ انواع شبکه‌های ارتباطی ایستا ۹
۷.۱ مدل PRAM برای محاسبات موازی ۱۱
۸.۱ مثالی از اجرای الگوریتم Sum-EREW با ورودی $n = ۸$. ۱۷
۹.۱ طرح کلی یک کامپیوتر تک هسته‌ای ۱۹
۱۰.۱ طرحی از یک تراشه پردازنده چندهسته‌ای ۲۰
۱۱.۱ فرآیند تک نخی و فرآیند چندنخی ۲۱
۱۲.۱ سیر تکامل پردازنده‌های چندهسته‌ای ۲۲
۱۳.۱ مقایسه ساده معماری‌های تک هسته‌ای، چندپردازنده‌ای و چندهسته‌ای ۲۳

۱.۲ مدل کامپیوتر ارائه شده توسط تسی و اتلا. ۲۸
۲.۲ مرتب‌سازی یک آرایه خطی با استفاده از یک آرایه سیستولیک. ۲۹
۳.۲ پیاده‌سازی یک رابطه h-تایی. ۳۸
۴.۲ عمل انتشار بسته ۴۰
۵.۲ عمل تجمیع بسته ۴۰
۶.۲ عمل انتشار کلی ۴۱

www.jozve.org

۷.۲ پیاده‌سازی عمل انتشار کلی در یک آرایه خطی در سطح سخت‌افزار. ۴۱

۸.۲ عمل تبادل کلی ۴۲

۹.۲ بیان روش کلی انجام عمل انتشار بسته با یک مثال ساده. ۴۳

۱۰.۲ عمل انتشار بسته یک‌تایی و نحوه برچسب‌گذاری. ۴۴

۱۱.۲ عمل کپی در حافظه محلی پردازنده $p_۱$ ۴۵

۱۲.۲ عمل انتشار بسته در $CGM\ (۲۰,۵)$ ۴۷

۱۳.۲ عمل انتشار کلی در $CGM\ (۹,۳)$ ۴۸

۱۴.۲ عمل تبادل کلی در $CGM\ (۹,۳)$ ۴۸

۱.۳ مقایسه دو مجموعه محدب و غیرمحدب ۵۳

۲.۳ تعریف غشای محدب ۵۴

۳.۳ روش تشخیص یک ضلع غشای فوقانی و مفهوم گردش به راست. ۵۴

۴.۳ نقطه c مغلوب $\overline{ab}$ است. ۵۸

۵.۳ افراز مجموعه S به سه زیر مجموعه ۵۹

۶.۳ روش تشخیص عنصر بعدی p ۶۰

۷.۳ شکل مثال۵.۳.۳. . . . . ۶۰

۸.۳ اثبات درستی رویه$QueryFindNext$، حالت اول. ۶۲

۹.۳ اثبات درستی رویه$QueryFindNext$، حالت دوم. ۶۲

۱۰.۳ عدم حضور نقاط مغلوب $\overline{x_i^*Next\left(x_i^*\right)}$ در غشای نهایی. ۶۴

۱۱.۳ اثبات درستی قسمت ۳ حالت دوم در شناسایی نقاط مغلوب. ۶۶

۱۲.۳ محاسبه عناصر بعدی غشای $X_۱$ در یک $CGM(۱۶,۴)$. ۶۷

۱۳.۳ مجموعه جداکننده G_i و مجموعه‌های R_i^- و R_i^+. ۶۸

۱۴.۳ انواع حالات ایجاد شده در صورت برقراری فرض خلف در لم ۱۰.۳.۳. . . . . ۷۰

۱۵.۳ عنصر x_i^* برابر با $lm\left(R_i \cup g_i^*\right)$ است. ۷۱

۱۶.۳ نحوه اجرای گام اول رویه $FindLMSubset$ بر روی $CGM\ (۱۶,۱۲)$. ۷۳

۱۷.۳ نحوه اجرای گام دوم رویه $FindLMSubset$ بر روی $CGM\ (۱۶,۱۲)$. ۷۴

www.jozve.org

۱۸.۳ عمل انتشار بسته مورد نیاز گام دوم رویه $FindLMSubset$. ۷۵

۱۹.۳ برچسب‌گذاری قلم‌های داده‌ای حافظه پردازنده q_1^3. ۷۶

۲۰.۳ نحوه اجرای گام چهارم رویه $FindLMSubset$ بر روی $CGM\ (16, 12)$، برای G_1. . . . . ۷۷

۲۱.۳ نحوه اجرای گام چهارم رویه $FindLMSubset$ در پردازنده‌های گروه Δ_4. . . . . ۷۸

۲۲.۳ نحوه محاسبه g_i^* در گام پنجم رویه $FindLMSubset$. ۷۸

۲۳.۳ محاسبه ترکیب p غشای فوقانی با فضای حافظه $\frac{n}{p} \geq p$ در هر پردازنده. ۸۵

۱.۴ p ثابت $(p = 32)$. ۹۴

۲.۴ n ثابت $(n = 1500000)$. ۹۵

۳.۴ نسبت $\frac{n}{p}$ ثابت $\left(\frac{n}{p} = 200000\right)$. ۹۵

۴.۴ زمان اجرا به تفکیک گام‌های مختلف الگوریتم $(p = 80, n = 800000)$. ۹۶

www.jozve.org

لیست الگوریتم‌ها

۱ $Sum - EREW$ ۱۶

۲ $Sum - MessagePassing$ ۱۸

۳ طرح کلی الگوریتم CGM ۳۳

۴ محاسبه ترتیبی غشای محدب ۵۵

۵ غشای فوقانی S ۵۷

۶ $MergeHulls۱\left(X_i\left(۱ \leq i \leq p\right), S, n, p, UH\right)$ ۸۱

۷ $MergeHulls۲\left(X_i, S, n, p, UH\left(S\right), \epsilon\right)$ ۸۸

www.jozve.org

www.jozve.org

لیست رویه‌ها

۱ $QueryFindNext\,(X, c, q)$ ۶۱

۲ $FindLMSubset\,(\Delta_i, k, w, G_i, g_i^*)$ ۷۲

۳ $BuildHulls\,(\Phi_i, \Psi_j, p, k, w, \epsilon)$ ۸۶

www.jozve.org

www.jozve.org

پیشگفتار

با گسترش روزافزون استفاده از کامپیوتر برای کارهای تحقیقاتی در دانشگاه‌ها، مراکز تحقیقاتی و شرکت‌های تجاری، نیاز به پردازش سریع‌تر افزایش یافته و به یک نیاز اساسی تبدیل شده است. امروزه پردازش موازی نقش بسیار جدی در مرتفع‌سازی این نیاز ایفا می‌کند.

سرعت کامپیوترهای شخصی کنونی نسبت به اجداد خود به طور سرسام‌آوری افزایش یافته است اما حتی این سرعت نجومی نیز در اجرای برخی از برنامه‌های پیشرفته، کند است. از جمله عرصه‌هایی که احتیاج به کامپیوترهایی با سرعت پردازش بسیار بالا دارند می‌توان به برنامه‌های شبیه‌سازی در تحقیقات هسته‌ای، فناوری نانو، برنامه‌های پیش‌بینی وضعیت هوا، برنامه‌های فیلم‌سازی کامپیوتری، برنامه‌های ساخت انیمیشن حرفه‌ای و بسیاری از زمینه‌های مختلف دیگر را نام برد که همگی به سرعت پردازش بسیار زیاد نیاز دارند تا در یک زمان مناسب به نتیجه برسند.

یک راه حل برای این معضل، استفاده از سوپرکامپیوترها است. درست است که سرعت پردازش سوپرکامپیوترها بسیار بالاتر از کامپیوترهای شخصی است اما استفاده از آنها در در همه موارد مقرون به صرفه نیست؛ ضمن آنکه این فناوری در انحصار بعضی از کشورهای توسعه‌یافته است و سایر کشورها از دسترسی به این تجهیزات استراتژیک محروم هستند.

راه حل دیگر در دستیابی به سرعت پردازش بسیار بالا استفاده از روش پردازش موازی است. به بیان ساده در این روش چند پردازنده معمولی با همکاری یکدیگر به اجرای یک برنامه می‌پردازند که طی این همکاری، برنامه با سرعت بالاتری اجرا می‌شود.

برای اینکه بتوان از امکانات و تجهیزات یک سیستم موازی به درستی استفاده نمود، نیاز به سیستم عامل‌هایی است که موازی سازی را پشتیبانی کنند، همچنین برای حل مسائلی که قابلیت موازی‌سازی را دارند، لازم است که الگوریتم‌ها و برنامه‌هایی موازی ارائه گردند. اولین گام در برنامه‌نویسی موازی این است که طراح برنامه،

www.jozve.org

بتواند به صورت موازی فکر کند، مسئله را به درستی شناخته و بر روی قسمت‌هایی از مسئله که قابلیت اجرای همزمان دارند، تمرکز کند. در این پایان‌نامه سعی شده است تا ملزومات و پیش‌نیازهای برنامه‌نویسی موازی معرفی شود و سپس به عنوان نمونه، برای حل یک مسئله هندسی، الگوریتم موازی ارائه می‌گردد.

در فصل اول مقدمه‌ای بر مفاهیم برنامه‌نویسی موازی بیان می‌شود. در فصل دوم مدل موازی مورد بحث این تحقیق معرفی می‌گردد و در فصل سوم، ابتدا مسئله هندسی محاسبه غشای محدب معرفی شده و سپس برای حل این مسئله هندسی، الگوریتمی موازی ارائه و تحلیل می‌شود.

www.jozve.org

فصل ۱

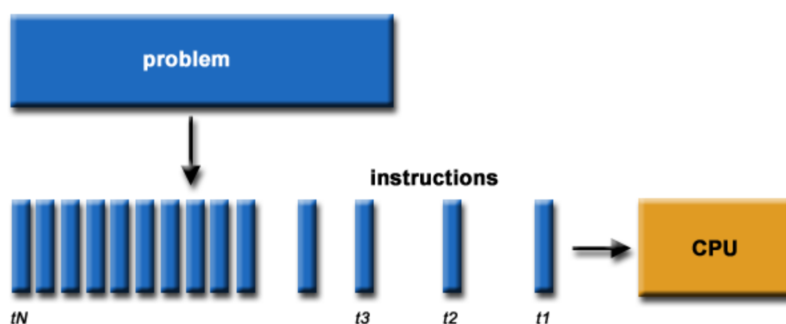
مقدمه‌ای بر برنامه‌نویسی موازی

www.jozve.org

این فصل مروری اجمالی بر مفهوم پردازش موازی، اهمیت و کاربردهای آن و همچنین مروری بر مفاهیم و ملزومات برنامه‌نویسی موازی خواهد بود. اکثر مفاهیمی که در ادامه فصل آورده شده از مراجع [۸، ۲۰، ۲۳] استخراج گردیده است. در انتهای فصل نیز، تکنولوژی چندهسته‌ای که در چند سال اخیر پیشرفت چشمگیری داشته است، معرفی می‌گردد.

۱.۱ محاسبه موازی

محاسبه موازی به معنای اجرای همزمان قسمت‌های مختلف یک برنامه در چند پردازنده به منظور حصول سریع‌تر نتایج است. در پردازش ترتیبی، دستورات به ترتیب در پردازنده اجرا می‌شوند و سرعت اجرا متناسب با سرعت پردازنده است (شکل ۱.۱). در پردازش موازی دستورات در چند پردازنده اجرا می‌شوند ولی سرعت اجرا الزاماً برابر با تعداد پردازنده‌ها ضربدر سرعت یک پردازنده نیست (شکل ۲.۱). محاسبه موازی در بخش‌های مختلف کامپیوتر اعم از سخت‌افزار و نرم‌افزار شکل می‌گیرد، بنابراین برای بررسی کلیات محاسبه موازی باید به جنبه‌های مختلف سخت‌افزاری و نرم‌افزاری آن پرداخت.

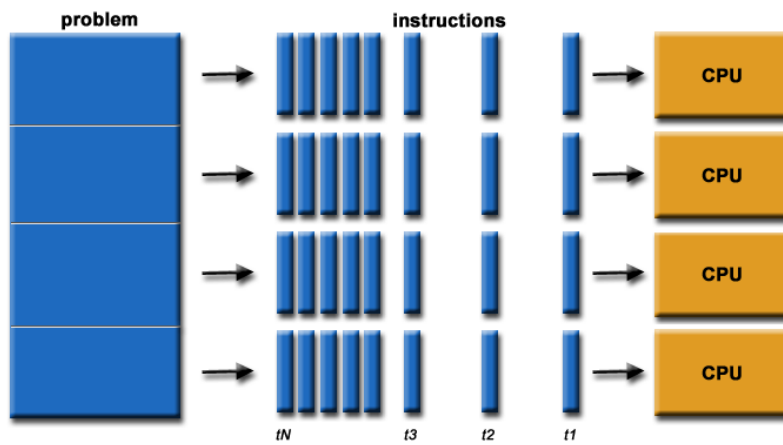


شکل ۱.۱: محاسبات سریال

از پردازش موازی در جهت افزایش قدرت کامپیوترها استفاده می‌شود. اما اصلی‌ترین استفاده از آن در حل مسائل و مدل‌های علمی و مهندسی است (شکل ۳.۱). از جمله این حوزه‌ها می‌توان به موارد زیر اشاره کرد:

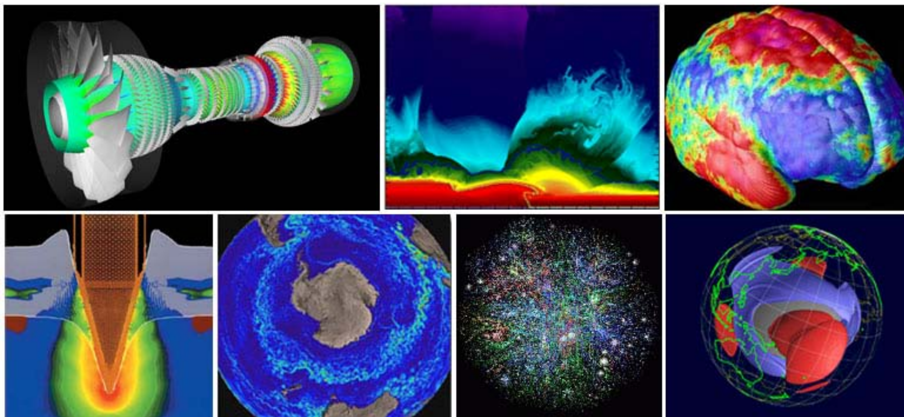
- فیزیک کاربردی، هسته‌ای، ذرات بنیادی، ماده چگال، گداخت هسته‌ای، فوتونیک و نانو
- اتمسفر، زمین و محیط زیست
- فناوری زیستی و ژنتیک
- زمین‌شناسی و زلزله‌شناسی

www.jozve.org



شکل ۲.۱: محاسبات موازی و کاربرد آن

- مهندسی مکانیک؛ از اندام مصنوعی تا مصنوعات فضایی
- مهندسی الکترونیک؛ طراحی مدار، میکروالکترونیک
- علوم کامپیوتر و ریاضی



شکل ۳.۱: تصاویری از داده‌های محاسباتی در رشته‌های مختلف

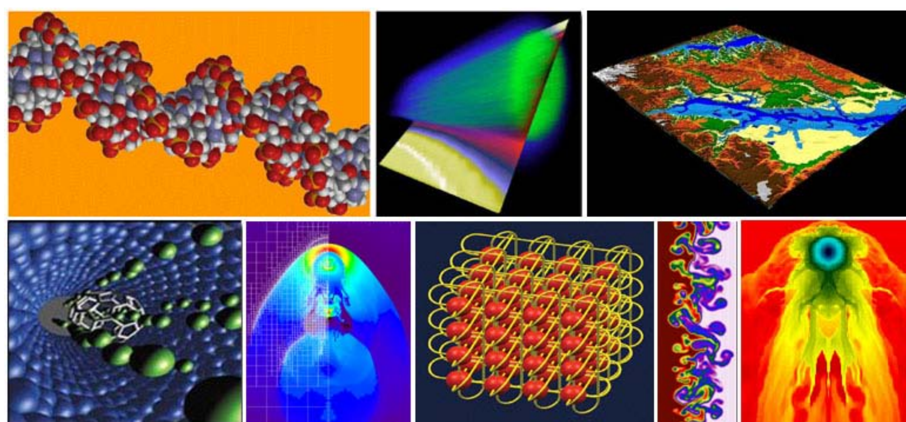
اما امروزه نه تنها حل مسایل علمی احتیاج به پردازش موازی دارد بلکه برخی از نرم‌افزارهای تجاری نیز به کامپیوترهای سریع نیاز دارند. بسیاری از این برنامه‌ها احتیاج به پردازش حجم زیادی از داده به شکل بسیار پیچیده دارند. از جمله این برنامه‌ها می‌توان به موارد زیر اشاره کرد:

- پایگاه‌های عظیم داده و عملیات داده کاوی[۱]
- اکتشاف نفت

[۱]Data Mining

www.jozve.org

- موتورهای جستجوی وب، سرویس‌های تجاری تحت وب
- تصویربرداری و تشخیص پزشکی (شکل ۴.۱)
- طراحی و شبیه‌سازی دارو
- مدیریت شرکت‌های ملی و چند ملیتی
- مدل‌سازی مالی و اقتصادی
- گرافیک پیشرفته و فناوری واقعیت مجازی[۲] [۹]
- فناوری چند رسانه‌ای و شبکه ویدیویی



شکل ۴.۱: تصاویر ایجاد شده توسط نرم‌افزارهای تجاری

دلایل اصلی استفاده از محاسبات موازی عبارتند از:

صرفه‌جویی در زمان و هزینه: استفاده از منابع بیشتر در یک کار، زمان انجام کار را کاهش می‌دهد. علاوه بر این، استفاده از چندین منبع ارزان به جای استفاده از یک منبع گران‌قیمت سبب کاهش هزینه‌ها می‌گردد.

حل مسائل بزرگتر: بسیاری از مسائل بزرگ و پیچیده که حل آن‌ها توسط یک کامپیوتر، خصوصاً در کامپیوترهای با حافظه محدود، غیرعملی یا غیرممکن است، با استفاده از کامپیوترهای موازی قابل حل می‌باشند. به عنوان مثال در موتورهای جستجوگر وب و پایگاه داده‌ها لازم است که میلیون‌ها تراکنش در یک ثانیه پردازش شوند.

---

[۲] Virtual Reality Technology

www.jozve.org

فراهم نمودن همزمانی: در منابع محاسباتی منفرد، در یک زمان تنها یک کار را می‌توان انجام داد، در حالی که در منابع محاسباتی چندگانه، چندین کار به طور همزمان قابل انجام است. به عنوان مثال AccessGrid[۳] یک شبکه همکاری جهانی است که مردم از سراسر جهان می‌توانند به طور همزمان و به صورت مجازی با یکدیگر ملاقات نموده و مراودات کاری خود را انجام دهند.

استفاده از منابع غیرمحلی: در مواقعی که منابع محاسباتی محلی برای حل مسئله مورد نظر، محدود هستند می‌توان به‌واسطه شبکه‌های گسترده و یا اینترنت از منابع غیرمحلی، برای حل مسئله کمک گرفت. به عنوان مثال در Folding@home[۴] که یک پروژه محاسبات توزیع‌شده است، غالب بر ۳۴۰۰۰۰ کامپیوتر متصل به اینترنت به کار گرفته می‌شود تا در مسئله تشخیص چین‌خوردگی‌های پروتئین، توان محاسباتی ۴.۲ ترافلاپس[۵] حاصل گردد (هر ترافلاپس معادل یک تریلیون عملیات ممیز شناور در ثانیه است).

۲.۱ معماری کامپیوتر موازی

پرطرفدارترین طبقه‌بندی معماری سیستم‌های کامپیوتری در سال ۱۹۶۶ توسط فلین[۶] تعریف شد. طرح طبقه‌بندی فلین[۷] براساس جریان اطلاعات صورت گرفته است. اطلاعاتی که پردازنده با آن سر و کار دارد را می‌توان به دو دسته دستورالعمل و داده تقسیم نمود. طبق طبقه‌بندی فلین جریان دستورالعمل یا داده می‌تواند به یکی از صورت‌های تکی یا چندتایی باشد. بر این اساس معماری سیستم‌های کامپیوتری را می‌توان به چهار دسته تقسیم نمود:

۱. جریان یک دستورالعمل، یک داده (SISD): این معماری برای یک کامپیوتر ترتیبی (غیر موازی) است. در این روش تنها یک جریان دستوری توسط یک پردازنده در طول هر پالس ساعت مورد عمل واقع می‌شود و همین‌طور یک جریان داده در طول پالس ساعت مورد استفاده قرار می‌گیرد. در این روش دستورها به طور قطعی انجام می‌شوند و وابسته به عمل پردازنده دیگر نیستند. کامپیوترهای ترتیبی متعارف

---

[۳] www.accessgrid.org

[۴] folding.stanford.edu

[۵] TeraFLOPS

[۶] Flynn

[۷] Flynn's Taxonomy

www.jozve.org

ون نیومن[8] در این دسته قرار می‌گیرند. این روش مورد استفاده در بیشتر کامپیوترها، از سیستم‌های Mainframe قدیمی گرفته تا کامپیوترهای شخصی امروزی است.

۲. جریان یک دستورالعمل، چند داده (SIMD): این معماری، یکی از انواع کامپیوترهای موازی است. آرایه‌های پردازنده‌ها[9] در این دسته قرار دارند. ماشین‌های SIMD یک واحد کنترل[10] دارند که مثل ماشین فن‌نیومن عمل می‌کند (یعنی یک دستور اجرا می‌کند) ولی بیش از یک عنصر پردازشگر[11] دارند. واحد کنترل سیگنال‌های کنترل را برای تمام عنصرهای پردازنده تولید می‌کند و این عناصر، یک عمل مشابه را در طول هر پالس ساعت، بر روی داده‌های متفاوت انجام می‌دهند. این روش برای حل مسایل خاص که از داده‌هایی با الگوی ثابت پیروی می‌کنند، مناسب است. از جمله این مسایل می‌توان به پردازش تصویر اشاره کرد.

۳. جریان چند دستورالعمل، یک داده (MISD): در این طرح موازی، یک جریان داده به چند واحد پردازش داده، ارسال می‌شود. هر واحد پردازش به طور مستقل با جریان‌های دستور مستقل روی داده‌ها عمل می‌کند. تاکنون تعداد کمی از نمونه‌های واقعی این کلاس از کامپیوترهای موازی وجود دارند. به عنوان نمونه، می‌توان به کامپیوتر آزمایشیC.mmp کارنگی ملون[12] اشاره کرد. از جمله کاربردهایی که برای این روش می‌توان مثال زد یکی، اعمال چند فیلتر فرکانسی روی یک جریان سیگنال و دیگری، اعمال چند الگوریتم رمزگذاری در باز کردن یک پیغام کد شده است.

۴. جریان چند دستورالعمل، چند داده (MIMD): به این دسته چندپردازنده نیز می‌گویند. در این معماری هر پردازنده امکان اجرای چند جریان دستوری جداگانه را دارد و این دستورات روی چند جریان داده مختلف اعمال می‌شود. سوپرکامپیوترهای امروزی، کامپیوترهای موازی خوشه‌ای[13]، کامپیوترهای

---

[8] Von Neumann

[9] Array processers

[10] Control unit

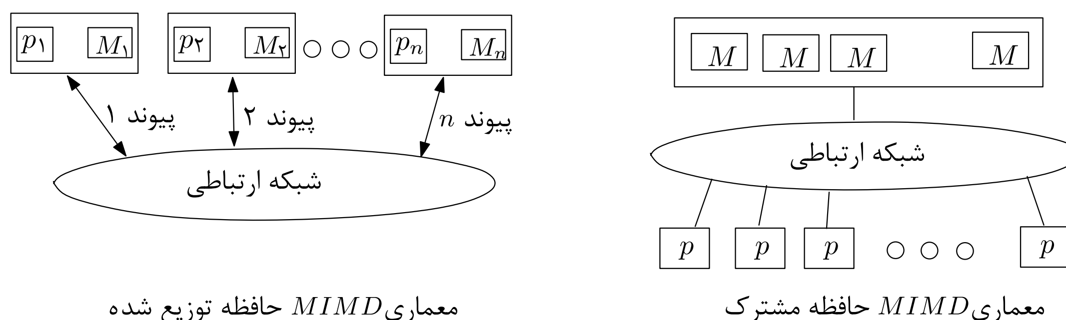
[11] Processing Element

[12] Carnegie-Mellon C.mmp

[13] Cluster parallel computers

چندپردازنده متقارن[۱۴] و کامپیوترهای چندهسته‌ای[۱۵] امروزی از این معماری استفاده می‌کنند. البته بیشتر کامپیوترهای با معماری MIMD از زیر مؤلفه‌های اجرایی SIMD بهره می‌برند.

یک ماشین MIMD (دسته چهارم)، دارای پردازنده‌هایی با واحد کنترل اختصاصی بوده و قادر است دستورالعمل‌های مختلف را بر روی داده‌های مختلف به صورت همزمان اجرا نماید. این روش معمول‌ترین طرح کامپیوتر موازی است و کامپیوترهای مدرن به سمت این معماری حرکت می‌کنند. این نوع معماری‌ها از چندین پردازنده و چندین ماژول حافظه که از طریق شبکه‌های ارتباطی به هم متصل شده‌اند، ساخته می‌شوند و به دو گروه اصلی تقسیم می‌گردند: حافظه مشترک و حافظه توزیع شده (انتقال پیام). شکل ۵.۱ ساختار کلی این دو گروه را نشان می‌دهد. در این شکل، p نشان‌دهنده پردازنده و M نشان‌دهنده ماژول‌های حافظه است.



شکل ۵.۱: انواع معماری‌های MIMD

۱.۲.۱ سیستم‌های حافظه مشترک

در سیستم‌های حافظه مشترک[۱۶]، تمام پردازنده‌ها یک حافظه سراسری مشترک دارند. ارتباط میان عملیات در حال اجرا روی پردازنده‌های مختلف با نوشتن و خواندن از حافظه سراسری برقرار می‌شود. همچنین هماهنگ‌سازی و همزمانی تمام پردازنده‌های میانی از طریق این حافظه مشترک انجام می‌شود. چنانچه تمامی پردازنده‌ها زمان دسترسی یکسانی به هر مکان حافظه داشته باشند، سیستم حافظه مشترک را سیستم

[۱۴] Symmetric Multiprocessor (SMP)

[۱۵] Multi core computers

[۱۶] Shared Memory

www.jozve.org

چندپردازنده متقارن می‌نامند. در طراحی سیستم‌های حافظه مشترک برخی مسائل اساسی نظیر کنترل دسترسی و وابستگی[۱۷] داده‌ها، همزمانی، حفاظت و امنیت وجود دارد که باید مورد توجه قرار گیرند [۲۰].

۲.۲.۱ سیستم‌های حافظه توزیع شده

سیستم‌های مبتنی بر حافظه توزیع شده[۱۸] (که به آن سیستم انتقال پیام نیز گفته می‌شود)، گروهی از چندپردازنده‌ها هستند که در آن هر پردازنده به حافظه محلی خود دسترسی دارد. برخلاف سیستم‌های حافظه مشترک، ارتباطات در این سیستم‌های از طریق دستورات ارسال و دریافت پیام صورت می‌گیرد که باید توسط برنامه‌نویس در نرم‌افزار کاربردی نوشته شود. یک گره در چنین سیستمی شامل یک پردازنده و حافظه محلی آن می‌باشد. هر گره معمولاً قابلیت ذخیره‌سازی پیام‌ها در میانگیر (مکان‌های حافظه موقتی که پیام‌ها در آنجا منتظر می‌مانند تا ارسال یا دریافت شوند) را دارد و عملیات ارسال/ دریافت را همزمان با پردازش انجام می‌دهد. پردازش پیام و انجام محاسبات به صورت همزمان توسط سیستم‌عامل موجود صورت می‌گیرد. سیستم‌های با حافظه توزیع‌شده از قابلیت توسعه بالایی برخوردار هستند و واحدهای پردازشگر آن‌ها را با روش‌های متنوع، از ساختارهای ارتباطی با معماری خاص گرفته تا شبکه‌های پراکنده جغرافیایی می‌توان به یکدیگر مرتبط نمود. قابلیت توسعه بدان معناست که تعداد پردازنده‌ها را بدون کاهش قابل توجه در کارایی، بتوان افزایش داد [۲۰] .

۳.۱ شبکه‌های ارتباطی داخلی

شبکه‌های ارتباطی سیستم‌های چندپردازنده[۱۹] را می‌توان براساس معیارهای مختلفی طبقه‌بندی نمود که یکی از این معیارها توپولوژی شبکه است. توپولوژی مشخص می‌کند که چگونه پردازنده‌ها و حافظه‌ها به پردازنده‌ها و حافظه‌های دیگر متصل می‌شوند. برای مثال در توپولوژی اتصال کامل هر پردازنده به تمامی پردازنده‌های دیگر موجود در سیستم‌های کامپیوتری متصل می‌شود. به صورت کلی توپولوژی شبکه‌های ارتباطی را می‌توان به دو گروه ایستا و پویا تقسیم نمود. در شبکه‌های ایستا تمامی ارتباط‌ها در زمان طراحی بدون توجه به این‌که مورد نیاز هستند یا نه ساخته می‌شود. در این شبکه‌ها پیام‌ها باید از پیوندهای معینی عبور کنند. شبکه‌های ارتباطی پویا در صورت نیاز اتصالاتی بین دو یا چند گره می‌سازند تا پیام‌ها از این پیوندها عبور کنند. توپولوژی‌های خطی،

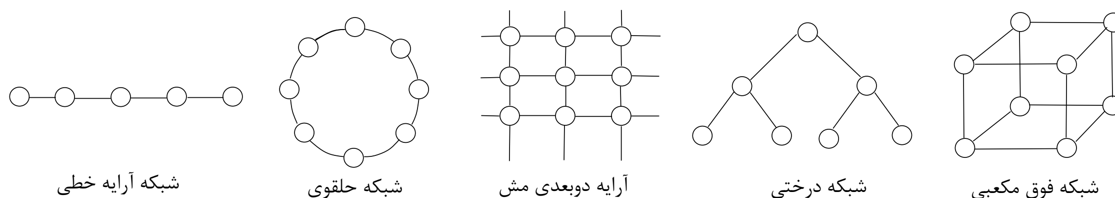
---

[۱۷] Coherency

[۱۸] Distributed Memory

[۱۹] Multiprocessors Interconnection Networks

حلقوی، مش[۲۰]، درختی و فوق مکعبی[۲۱] که طرح کلی آن‌ها در شکل ۶.۱ نمایش داده شده است، نمونه‌هایی از شبکه‌های ارتباطی ایستا هستند و شبکه‌های تک‌باس[۲۲]، چندباس[۲۳]، کراس‌بار[۲۴] و چندلایه‌ای[۲۵] نمونه‌هایی از شبکه‌های ارتباطی پویا هستند. برای توضیحات بیشتر درباره انواع توپولوژی‌های شبکه پردازنده‌ها و فاکتورهای ارزیابی کیفیت هر یک، مراجع [۸، ۲۰، ۲۳] را ببینید.



شکل ۶.۱: انواع شبکه‌های ارتباطی ایستا

۴.۱ مدل‌های برنامه‌نویسی موازی

مدل‌های تجریدی به علت طبیعت آرمان‌گرای آن‌ها ممکن است در دنیای واقعی مناسب به نظر نرسند. اما ماشین‌های تجریدی در مطالعه الگوریتم‌های موازی توزیع شده و ارزیابی کارایی پیش‌بینی شده، مستقل از ماشین‌های موازی بسیار مفیدند.

بدیهی است که اگر اجرای یک الگوریتم بر روی یک سیستم تجریدی رضایت‌بخش نباشد، پیاده‌سازی آن بر روی یک سیستم واقعی بی‌معنی است. با وجود این‌که مدل‌های تجریدی برخی ملاحظات عملی در سیستم‌های واقعی موازی و توزیع‌شده را در نظر نمی‌گیرند، دشواری یافتن محدودیت‌های اجرایی و برآوردهای پیچیدگی را کاهش می‌دهند. بنابراین الگوریتم‌های موازی همواره طبق یک مدل انتخابی طراحی شده و سپس تلاش می‌شود تا مدل به یک برنامه قابل اجرا تبدیل گردد. برای اجرایی کردن یک مدل در دنیای واقعی مجموعه‌ای

[۲۰] Mesh

[۲۱] Hypercube

[۲۲] Single Bus

[۲۳] Multiple Bus

[۲۴] Crossbar

[۲۵] Multistage

www.jozve.org

از زبان‌ها، کامپایلرها، کتابخانه‌ها، سیستم‌های ارتباطی و ورودی- خروجی موازی به کار گرفته می‌شود. تاکنون مدل‌های موازی مختلفی نظیر حافظه مشترک، داده-موازی، رشته‌ها، انتقال پیام و غیره ارائه شده است که در ادامه به توصیف دو مدل رایج پرداخته می‌شود.

۱.۴.۱ مدل حافظه مشترک

در مدل‌های حافظه مشترک، یک برنامه موازی به قسمت‌های مختلفی تقسیم شده که به هر قسمت وظیفه[۲۶] گفته می‌شود. اجرای هر یک از وظایف به یک پردازنده محول شده و تمام پردازنده‌ها بر روی داده‌های ذخیره شده در حافظه مشترک عمل می‌کنند. از مکانیزم‌های همزمانی مختلفی نظیر قفل‌ها[۲۷] و سمافورها[۲۸]برای کنترل دسترسی همزمان پردازنده‌ها به خانه‌های حافظه استفاده می‌شود. در تحلیل الگوریتم‌های موازی نوشته شده در این مدل، معیارهایی نظیر زمان اجرا، تعداد پردازنده‌هایی که الگوریتم برای حل مسئله به کار می‌برد و هزینه الگوریتم موازی که حاصل‌ضرب زمان اجرا در تعداد پردازنده‌هاست مورد توجه قرار می‌گیرد.
یکی از مدل‌هایی که در سیستم‌های حافظه مشترک به طور گسترده مورد استفاده قرار می‌گیرد، مدل ماشین دسترسی تصادفی موازی[۲۹](PRAM) است. این مدل در سال ۱۹۸۷ توسط فورچن[۳۰] و وایلی[۳۱] برای مدل‌سازی کامپیوترهای موازی آرمانی ارائه گردید. یک PRAM متشکل از یک واحد کنترل و یک حافظه سراسری است که برای p پردازنده مشترک است. علاوه بر حافظه سراسری که ارتباط پردازنده‌ها از طریق آن برقرار می‌شود، هر پردازنده به منظور کاهش مراجعات به حافظه مشترک و تسریع پردازش، حافظه اختصاصی مربوط به خود را دارد. شکل ۷.۱ دیاگرامی را نشان می‌دهد که اجزای این مدل را شرح می‌دهد. در این مدل فرض می‌شود که به تعداد دلخواه پردازنده در اختیار هست و هیچ دو پردازنده‌ای مستقیماً به یکدیگر متصل نیستند و ارتباطات تنها از طریق خواندن و نوشتن در حافظه مشترک انجام می‌شود. حالت‌های مختلفی برای عملیات خواندن و نوشتن در حافظه مشترک وجود دارد که بر این اساس، می‌توان PRAM را به کلاس‌های زیر تقسیم نمود:

---

[۲۶] Task

[۲۷] قفل‌ها روش‌های همزمانی هستند که سیستم‌عامل به کار می‌گیرد تا مانع استفاده همزمان چندین وظیفه از یک منبع گردد.
[۲۸] سمافور رایج‌ترین مکانیزم پیاده‌سازی یک قفل است و به متغییری گفته می‌شود که در محیط‌های همروند برای کنترل دسترسی فرآیندها به منابع مشترک به کار می‌رود. سمافور می‌تواند به دو صورت دودویی و یا شمارنده اعداد صحیح باشد.

[۲۹] Parallel Random Access Machine

[۳۰] Fortune

[۳۱] Wylie

www.jozve.org

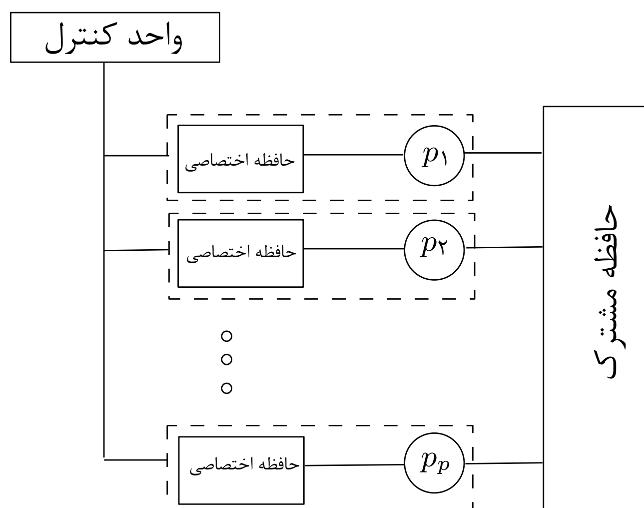
– خواندن انحصاری، نوشتن انحصاری (EREW)[۳۲]: دسترسی‌های خواندن و نوشتن در یک مکان حافظه، انحصاری است.

– خواندن انحصاری، نوشتن همزمان (ERCW)[۳۳]: چند پردازنده اجازه نوشتن همزمان در یک مکان حافظه را دارند ولی دسترسی خواندن انحصاری است.

– خواندن همزمان، نوشتن انحصاری (CREW)[۳۴]: اجازه دسترسی‌های همزمان خواندن داده می‌شود ولی دسترسی‌های نوشتن انحصاری است.

– خواندن همزمان، نوشتن همزمان (CRCW)[۳۵]: دسترسی‌های خواندن و نوشتن همزمان مجاز است.

برای برنامه‌نویسی در این مدل اغلب یک واسط برنامه‌نویسی به نام OpenMP استفاده می‌شود که برای زبان‌های C/C++ و فرترن به‌کار برده می‌شود. سیستم‌عامل‌های اصلی که این واسط بر روی آن‌ها قابل اجراست لینوکس، یونیکس و ویندوز NT می‌باشند.



شکل ۷.۱: مدل PRAM برای محاسبات موازی

[۳۲] Exclusive-Read Exclusive-Write

[۳۳] Exclusive-Read Concurrent-Write

[۳۴] Concurrent-Read Exclusive-Write

[۳۵] Concurrent-Read Concurrent-Write

۲.۴.۱ مدل انتقال پیام

مدل انتقال پیام شامل مجموعه‌ای از پردازنده‌هاست که هر یک تنها حافظه محلی مختص به خود را دارند و پردازنده‌ها با ارسال و دریافت پیام با یکدیگر ارتباط برقرار می‌کنند. انتقال داده بین پردازنده‌ها نیازمند عملیات متقابل بین پردازنده‌ها است به این معنی که هر عمل ارسال متناظر با یک عمل دریافت داده است. این مدل به دلیل مزیت‌های گوناگونی که دارد به طور گسترده در زمینه محاسبات موازی مورد استفاده قرار می‌گیرد. برخی از این مزیت‌ها عبارتند از:

- سازگاری با سخت‌افزار: این مدل برای استفاده در سوپرکامپیوترها و کلاسترها که شامل پردازنده‌های مجزایی هستند که توسط شبکه‌های ارتباطی به یکدیگر متصل‌اند، مناسب است.

- ساختار تابعی: مدل انتقال پیام یک مجموعه کاملی از توابع را ارائه می‌دهد که امکان کنترل‌هایی را فراهم می‌کند که در مدل‌های دیگر نظیر مدل داده-موازی وجود ندارد.

- کارایی: استفاده موثر از پردازنده‌های مدرن امروزی، نیاز به مدیریت قوی بر سلسه مراتب حافظه، به‌ویژه حافظه‌های کش دارد. این مدل با اعطای ابزار صریح کنترل به برنامه‌نویس، امکان مدیریت مکان داده‌ها را فراهم می‌آورد.

مشکل اصلی این روش این است که مسئولیت آنچه اتفاق می‌افتد بر عهده برنامه‌نویس است. یعنی برنامه‌نویس باید صریحاً با فراخوانی توابع موجود، داده‌ها را بین همه پردازنده‌ها توزیع کرده و همزمانی و ارتباط بین پردازنده‌ها را مدیریت کند.

مدل‌های مختلفی از انتقال پیام نظیر MPI[۳۶]، BSP، CGM و غیره ارائه گردیده است که هر یک کتابخانه‌های مخصوص به خود را دارند. مولفه‌های رایج کتابخانه‌های مدل انتقال پیام شامل موارد زیر است:

- روال‌های مدیریت پردازنده که وظیفه مقداردهی اولیه و نهایی پردازنده‌ها، تعیین تعداد پردازنده و شناسه‌های پردازنده‌ها را برعهده دارند.

- روال‌های ارتباطات نقطه به نقطه[۳۷] که ارسال و دریافت داده بین دو پردازنده معین را مدیریت می‌کند.

[۳۶] Message Passing Interface

[۳۷] Point-to-Point

www.jozve.org

– روال‌های گروه‌بندی و ارتباطات جمعی که عملیات تجمیع، انتشار داده‌ها و هماهنگی بین پردازنده‌ها را مدیریت می‌کنند.

در تحلیل الگوریتم‌های موازی نوشته شده در مدل انتقال پیام، پیچیدگی زمانی اجرای الگوریتم بر روی هر پردازنده و همچنین تعداد و زمان ارتباطات مورد نیاز مورد بررسی قرار می‌گیرد.

۵.۱ الگوریتم‌های موازی

بسیاری از الگوریتم‌ها به منظور استفاده از سخت‌افزار موازی باید دوباره طراحی شوند. برنامه‌هایی که در یک سیستم تک پردازنده درست کار می‌کنند ممکن است در یک محیط موازی هرگز کار نکنند. این بدان علت است که چند کپی از یک برنامه ممکن است با یکدیگر تداخل کنند (به طور نمونه تداخل در دسترسی همزمان به یک محل از حافظه). بنابراین نیاز اصلی یک سیستم موازی، برنامه‌نویسی دقیق مختص به خود است.
طراحی و توسعه برنامه‌های موازی اغلب یک فرآیند دستی محسوب می‌شود. برنامه‌نویس مسئول تعیین و اجرای واقعی موازی‌سازی است. توسعه‌دستی کدهای موازی اغلب فرآیندی زمان‌بر، پیچیده، تکراری و مستعد خطاست. در سال‌های اخیر بسیاری از سیستم‌های نرم‌افزاری جهت برنامه‌نویسی کامپیوترهای موازی طراحی شده‌اند که به برنامه‌نویس در تبدیل برنامه سریال به برنامه موازی کمک می‌کنند. این سیستم‌ها هم در سطح سیستم‌عامل و هم در سطح زبان‌های برنامه نویسی وجود دارند. آن‌ها باید ساز و کاری جهت تقسیم یک مسئله به چند وظیفه و تخصیص این وظایف به پردازنده‌ها داشته باشند. چنین ساز و کارهایی می‌تواند شامل موازی‌سازی ضمنی و یا موازی‌سازی صریح باشد.
در روش موازی‌سازی ضمنی، سیستم (که می‌تواند کامپایلر یا برنامه‌های دیگر باشد) به طور خودکار مسئله را به چند وظیفه تقسیم کرده و هر یک را به پردازنده‌ای اختصاص می‌دهد اما در روش موازی‌سازی صریح، برنامه‌نویس شخصاً مسئله را به چند وظیفه تفکیک و هر یک را به پردازنده‌ای ارجاع می‌دهد. روش موازی‌سازی ضمنی محدود به زیرمجموعه‌ای از کدها (اغلب حلقه‌های تکرار) است و انعطاف‌پذیری کمتری نسبت به موازی‌سازی صریح دارد. همچنین ممکن است نتایج اشتباهی تولید کند و کارایی را کاهش دهد. از این‌رو اغلب برنامه‌نویسی‌های موازی به صورت صریح انجام می‌گیرد.

طراحی الگوریتم‌های موازی

اولین گام در طراحی الگوریتم‌های موازی این است که برنامه‌نویس یاد بگیرد چگونه موازی فکر کند. قسمت‌هایی

www.jozve.org

از مسئله که قابلیت موازی‌سازی دارند را تشخیص داده و پس از انتخاب مدل، بدون توجه به ملزومات اجرایی، ذهن خود را صرفاً بر روی ارائه بهترین الگوریتم موازی متمرکز کند.

در حل یک مسئله به روش موازی نکات مختلفی باید در نظر گرفته شود. قبل از هر چیز باید مشخص گردد که آیا مسئله واقعاً قابلیت موازی‌سازی دارد؟ به عنوان مثال مسئله ساخت دنباله فیبوناتچی به علت ماهیت وابستگی داده‌ها، یک مسئله غیرموازی است. پس از این مرحله باید نقاط اصلی[۳۸] مسئله که بیشترین محاسبات در آن قسمت انجام می‌شود، شناسایی شده و حتی‌المقدور بر روی موازی‌سازی نقاط اصلی برنامه تمرکز شود. همچنین تنگناهای[۳۹] مسئله باید شناسایی شوند یعنی نقاطی که به علت وابستگی داده‌ها و یا نیاز به عملیات ورودی- خروجی، عمل موازی‌سازی متوقف می‌شود.

پس از این مرحله نوبت به تجزیه مسئله به بخش‌های مختلف است که هر بخش می‌تواند به عنوان یک وظیفه به یک پردازنده محول گردد. دو روش اساسی برای تقسیم کارهای محاسباتی بین پردازنده‌ها وجود دارد: تجزیه دامنه‌ای و تجزیه تابعی. در تجزیه دامنه‌ای داده‌های مسئله تقسیم شده و هر پردازنده دستورات یکسانی را بر روی داده‌های مربوط به خود اجرا می‌کند. در تجزیه تابعی دستورات محاسباتی که باید اجرا گردند بین پردازنده‌ها تقسیم می‌شوند. پس از تجزیه مسئله به وظایف مختلف، چنانچه نیاز به ارتباط بین وظایف باشد باید از رویه‌های همزمانی و عملیات ارتباطی بین پردازنده‌ها استفاده گردد.

۶.۱ معیار ارزیابی سیستم موازی

یکی از معیارهای ارزیابی سیستم موازی ضریب **تسریع**[۴۰] یا $S\left(p\right)$ است که به صورت زیر تعریف می‌شود: نسبت زمان مورد نیاز برای حل مسئله مورد نظر توسط یک پردازنده که با $T_{sequential}$ نشان داده می‌شود، به زمان مورد نیاز برای حل همان مسئله توسط یک سیستم موازی که از p پردازنده تشکیل شده است. زمان سیستم موازی با $T_{parallel}$ نشان داده می‌شود.

$$S\left(p\right) = \frac{T_{sequential}}{T_{parallel}}$$

---

[۳۸] Hotspots

[۳۹] Bottlenecks

[۴۰] Speed Up

www.jozve.org

چنانچه $T_{parallel} = O(\frac{T_{sequential}}{p})$ باشد، ضریب تسریع برابر p است و سیستم موازی بهینه است. در عمل، دستیابی به افزایش خطی سرعت (افزایش سرعت متناسب با تعداد پردازنده‌ها) بسیار مشکل است. این مشکل ناشی از طبیعت ترتیبی بسیاری از الگوریتم‌ها است به طوری‌که قسمت‌هایی از یک الگوریتم قابل موازی‌سازی و قسمت‌هایی غیر قابل موازی‌سازی است. بر طبق **قانون آمدال**[۴۱]، میزان تسریع یک الگوریتم موازی نسبت به یک الگوریتم ترتیبی به تعداد پردازنده‌های مورد استفاده بستگی ندارد بلکه به قسمتی از الگوریتم که قابل موازی‌سازی نیست وابسته است. اگر F برابر کسری از الگوریتم باشد که غیر قابل موازی‌سازی است و می‌بایست حتماً به صورت ترتیبی اجرا شود، ضریب تسریع، طبق قانون آمدال به صورت زیر تعریف می‌شود:

$$S(p) = \frac{۱}{F + \frac{۱-F}{p}}$$

فرض کنید که ده درصد از یک الگوریتم قابلیت موازی‌سازی ندارد یعنی $F = ۱۰\%$ اما بقیه الگوریتم به طور موازی توسط ۲۰ پردازنده اجرا می‌شود. در این حالت سرعت اجرای برنامه (نسبت به زمانی که تنها روی یک پردازنده اجرا شود) ۲۰ برابر نمی‌شود بلکه مطابق قانون آمدال ضریب تسریع تقریباً ۷ برابر می‌شود.

$$S(۲۰) = \frac{۱}{۰/۱ + \frac{۱-۰/۱}{۲۰}} \simeq ۷.$$

معیار دیگری که در ارزیابی سیستم‌های موازی به کار می‌رود **کارایی**[۴۲] یا $E(p)$ است که برابر با نسبت هزینه اجرای یک الگوریتم در سیستم ترتیبی به هزینه اجرای همان الگوریتم در یک سیستم موازی که از p پردازنده تشکیل شده است. هزینه اجرای یک الگوریتم برابر حاصل‌ضرب زمان اجرا در تعداد پردازنده‌های مورد استفاده است.

$$E(p) = \frac{۱ \times T_{sequential}}{p \times T_{parallel}} = \frac{S(p)}{p}$$

۷.۱ دو نمونه از الگوریتم موازی

در این بخش برای مسئله ساده محاسبه جمع n عدد، الگوریتمی موازی در دو مدل حافظه مشترک و انتقال پیام ارائه می‌شود که با استفاده از مراجع [۸، ۲۰] تنظیم شده است. این الگوریتم‌ها لزوماً بهینه نیستند و صرفاً جهت نمایش نحوه تعامل پردازنده‌ها در مدل بیان شده‌اند.

---

[۴۱] Amdahl's law

[۴۲] Efficiency

www.jozve.org

۱.۷.۱ الگوریتم محاسبه حاصل جمع آرایه‌ای از اعداد در مدل $EREW PRAM$

فرض کنید آرایه $A[1 \cdots n]$ در حافظه مشترک ذخیره شده است. هدف محاسبه حاصل جمع این اعداد به کمک $\frac{n}{2}$ پردازنده است. از آنجایی‌که نیازی به انجام چند عملیات همزمان خواندن و نوشتن روی یک مکان حافظه نیست، حالت EREW انتخاب می‌شود. جمع کل در نهایت در مکان $A[n]$ ذخیره می‌شود. برای سادگی فرض کنید n توان صحیحی از ۲ است. الگوریتم در $\log n$ تکرار به شرح زیر تکمیل خواهد شد. در تکرار اول تمامی پردازنده‌ها فعال هستند. در تکرار دوم تنها نیمی از پردازنده‌ها فعال خواهند بود و بقیه تکرارها هم به همین ترتیب خواهند بود. جزئیات تشریح شده در الگوریتم ۱ را ببینید.

شکل ۸.۱ نحوه اجرای الگوریتم را بر روی آرایه‌ای با هشت عنصر نشان می‌دهد. برای جمع این ۸ عنصر

الگوریتم ۱: $Sum - EREW$

ورودی: آرایه $A[1 \cdots n]$ که در حافظه مشترک ذخیره شده است.

خروجی: حاصل جمع عناصر آرایه A

برای مقادیر $i = 1$ تا $i = \log n$ محاسبات زیر را تکرار کن:

۱. برای همه پردازنده‌های p_j که $1 \leq j \leq \frac{n}{2}$ است، به صورت موازی بررسی کن:

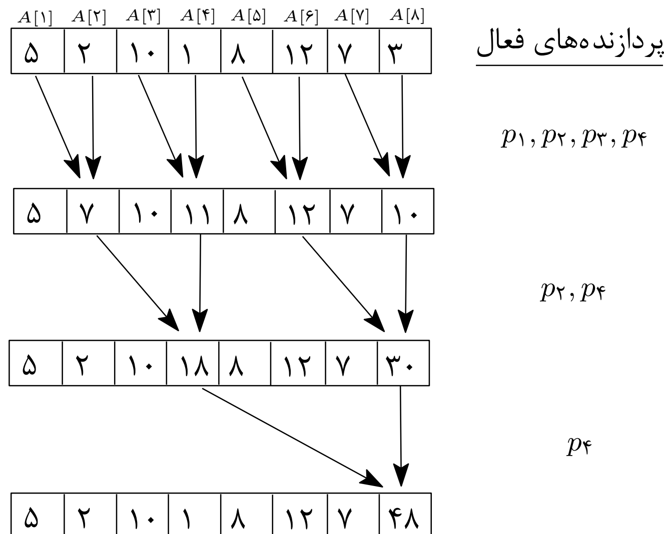
(آ) اگر شرط $\left(2j \mod 2^i = 0\right)$ برقرار است، آنگاه جایگذاری زیر را انجام بده:

$$A[2j] \longleftarrow A[2j] + A[2j - 2^{i-1}]$$

سه تکرار به شرح زیر مورد نیاز است. در اولین تکرار، پردازنده‌های p_1، p_2، p_3 و p_4 به ترتیب مقادیر ذخیره شده در مکان‌های ۱، ۳، ۵ و ۷ را با اعداد ذخیره شده در مکان‌های ۲، ۴، ۶ و ۸ جمع می‌کنند. در تکرار دوم پردازنده‌های p_2 و p_4 به ترتیب مقادیر ذخیره شده در مکان‌های ۲ و ۶ را با اعداد ذخیره شده در مکان‌های ۴ و ۸ جمع می‌کنند. در نهایت، در تکرار سوم پردازنده p_4 مقدار ذخیره شده در مکان ۴ را با مقدار ذخیره شده در مکان ۸ جمع می‌کند. بنابراین در نهایت مکان ۸ حاوی مجموع اعداد آرایه خواهد بود.

تحلیل پیچیدگی الگوریتم ۱: دستور شماره یک به تعداد $\log n$ بار اجرا می‌شود و هر تکرار زمان پیچیدگی ثابتی دارد. از این‌رو زمان اجرای الگوریتم $\log n$ می‌باشد. معیارهای پیچیدگی الگوریتم به صورت زیر خلاصه می‌شود:

www.jozve.org



شکل ۸.۱: مثالی از اجرای الگوریتم Sum-EREW با ورودی $n = ۸$.

زمان اجرا: $T(n) = O(\log n)$ تعداد پردازنده‌ها: $p = \frac{n}{۲}$

هزینه اجرا: $C(n) = O(n \log n)$ تسریع: $S(n) = \frac{n}{\log n}$

کارایی: $E(n) = \frac{۱}{\log n}$

یک الگوریتم ترتیبی مناسب می‌تواند حاصل جمع لیستی از n عنصر را در $O(n)$ محاسبه کند، بنابراین الگوریتم موازی ارائه شده زمان اجرا را کاهش می‌دهد ولی دارای هزینه بهینه نیست و تنها برای زمانی که تعداد پردازنده‌های موجود از $O(n)$ باشد، مناسب است. به چنین سیستم‌های موازی که تعداد پردازنده‌ها تقریباً از مرتبه داده‌های ورودی است و هر پردازنده به طور میانگین، تقریباً $O(۱)$ داده را پردازش می‌کند، به اصطلاح سیستم‌های **دانه‌ریز**[۴۳] گویند. در فصل ۲ توضیحات بیشتری در خصوص مفهوم دانه‌بندی ارائه می‌شود.

۲.۷.۱ الگوریتم محاسبه حاصل جمع آرایه‌ای از اعداد در مدل انتقال پیام

فرض کنید در یک سیستم موازی، p پردازنده موجود است و عناصر آرایه ورودی $A[۱ \cdots n]$ به طور مساوی بین هر یک از این پردازنده‌ها توزیع شده باشد که در این صورت هر پردازنده تعداد $\frac{n}{p}$ عنصر را در حافظه محلی خود ذخیره دارد. الگوریتم ۲ نحوه محاسبه حاصل جمع این اعداد در مدل انتقال پیام را نشان می‌دهد. در الگوریتم حافظه مشترک بخش ۱.۷.۱، هر پردازنده به هر مکان حافظه مشترک دسترسی داشته و در نتیجه می‌تواند از مجموع حاصل در زمان ثابت، مطلع شود. اما در روش انتقال پیام، مجموع نهایی تنها در حافظه

[۴۳] Fine Grained

www.jozve.org

محلی یکی از پردازنده‌ها وجود دارد و به منظور اطلاع سایر پردازنده‌ها از حاصل‌جمع، نتیجه به تمام پردازنده‌ها ارسال می‌گردد.

تحلیل پیچیدگی: گام‌های اول و سوم الگوریتم به صورت ترتیبی و به ترتیب در زمان‌های $\Theta\left(\frac{n}{p}\right)$ و $\Theta\left(p\right)$

الگوریتم ۲ : $Sum - MessagePassing$

ورودی: آرایه $A\left[1 \cdots n\right]$ که عناصر آن به طور مساوی بین هر یک از پردازنده‌ها توزیع شده است.

خروجی: حاصل جمع عناصر آرایه A

۱. هر پردازنده p_i، مجموع عناصر موجود در حافظه محلی خود را با استفاده از یک الگوریتم ترتیبی محاسبه می‌کند.

۲. هر پردازنده p_i، مجموع حاصل را که با m_i نشان داده می‌شود به پردازنده p_1 ارسال می‌کند.

۳. پردازنده p_1 مجموع m_i، $1 \leq i \leq p$ عنصر دریافتی را با استفاده از یک الگوریتم ترتیبی محاسبه می‌کند. این مجموع با M نشان داده می‌شود.

۴. پردازنده p_1 به منظور اطلاع سایر پردازنده‌ها از حاصل‌جمع نهایی، عنصر M را به تمام پردازنده‌ها ارسال می‌کند.

محاسبه می‌شوند. گام‌های دوم و چهارم الگوریتم گام‌های ارتباطی بوده و هزینه اجرای آن به نوع الگوریتم انتقال پیام و نوع شبکه ارتباطی بستگی دارد. توضیحات بیشتر درباره انواع عملیات ارتباطی و نحوه پیاده‌سازی آن‌ها، در فصل۲ ارائه می‌شود.

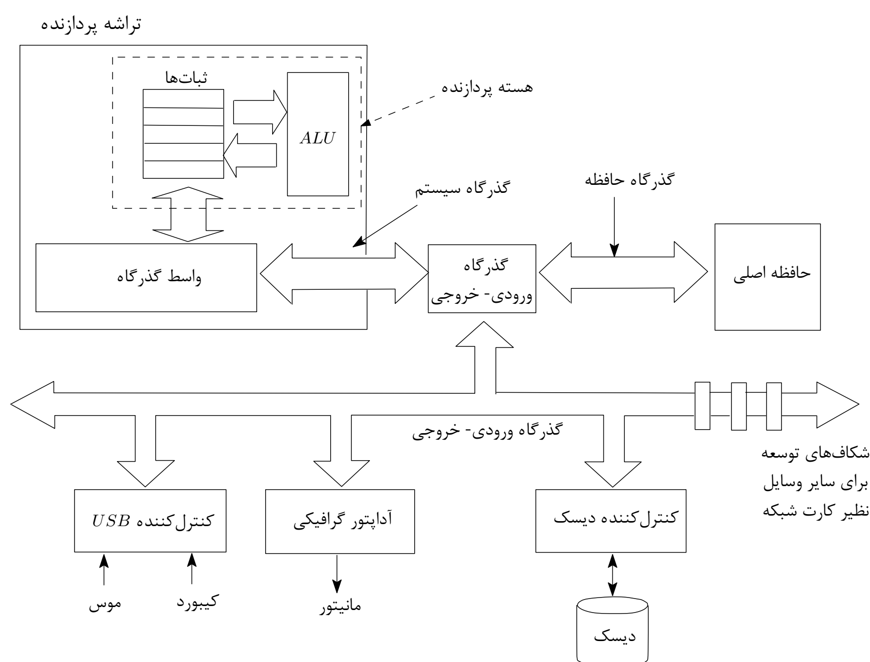
به سیستم‌های موازی‌ایی که تعداد پردازنده‌ها به مراتب کمتر از تعداد داده‌های ورودی است و هر پردازنده داده‌های زیادی را پردازش می‌کند، اصطلاحاً سیستم‌های **دانه‌درشت**[۴۴] گویند. در فصل ۲ توضیحات بیشتری در خصوص مفهوم دانه‌بندی ارائه می‌شود.

[۴۴] Coarse Grained

www.jozve.org

۸.۱ تکنولوژی چندهسته‌ای

پردازنده فیزیکی یا همان واحد پردازش مرکزی (CPU)[۴۵] اصلی‌ترین بخش یک سیستم‌کامپیوتری محسوب می‌شود و از قطعات مختلف سخت‌افزاری شبیه گذرگاه‌ها، واحد کنترل، واحد حساب و منطق(ALU)[۴۶]، حافظه کش و مجموعه‌ای از ثبات‌ها شامل ثبات‌های همه‌منظوره، ثبات‌های وضعیت پردازنده، ثبات‌های کنترل وقفه و غیره تشکیل شده است. واحد حساب و منطق و ثبات‌های کنترلی در واقع بخش اصلی یک پردازنده یا **هسته**[۴۷] پردازنده را تشکیل می‌دهند. شکل ۹.۱ طرح کلی یک کامپیوتر تک هسته‌ای را نشان می‌دهد. در یک پردازنده چندهسته‌ای، چندین واحد حساب و منطق بر روی یک قطعه نازک مستطیل شکل[۴۸] که از جنس نیمه‌هادی سیلیکان است، قرار می‌گیرند و از طریق واسط گذرگاه به یکدیگر متصل می‌شوند. شکل ۱۰.۱ طرحی از یک پردازنده چندهسته‌ای را نشان می‌دهد.



شکل ۹.۱: طرح کلی یک کامپیوتر تک هسته‌ای

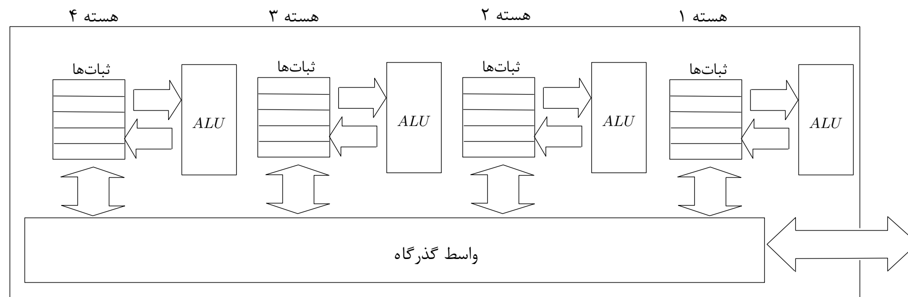
[۴۵] Central Processing Unit

[۴۶] Arithmetic Logic Unit

[۴۷] Core

[۴۸] Die

www.jozve.org



شکل ۱۰.۱: طرحی از یک تراشه پردازنده چندهسته‌ای

1.8.1 تکامل تکنولوژی چندهسته‌ای

قبل از بررسی سیر تکامل پردازنده‌های چندهسته‌ای، لازم است که مفهوم نخ[۴۹] معرفی شود. یک **نخ** کوچترین واحد اجرایی است که توسط سیستم‌عامل می‌تواند زمان‌بندی شود و در واقع واحد اصلی‌ای است که پردازنده را به کار می‌گیرد. هر نخ شامل اطلاعاتی نظیر شمارنده برنامه که به دستور جاری در برنامه اشاره می‌کند، همچنین شامل اطلاعات وضعیت پردازنده و منابع دیگری مانند پشته، متغیرهای محلی و سراسری مورد نیاز جهت اجراست. یک فرآیند یا برنامه در حال اجرا می‌تواند تک‌نخی[۵۰] (شکل ۱۱.۱ قسمت (الف)) و یا چندنخی[۵۱] باشد. همانطور که در شکل ۱۱.۱ قسمت (ب) می‌بینید نخ‌های یک فرآیند چندنخی، در منابعی مانند کد برنامه، داده و فایل‌های مورد نیاز برنامه مشترک هستند اما هر یک جهت اجرا، پشته، متغیرها و ثبات‌های مختص به خود را دارند. برای توضیحات بیشتر در خصوص فرآیند و نخ و نحوه مدیریت آن‌ها توسط سیستم‌عامل، به مراجع [۳۳، ۳۴] رجوع کنید.

در برنامه‌نویسی موازی دو واژه همروند[۵۲] و موازی متفاوت هستند. وقتی چندین نخ اجرایی به صورت **موازی** در حال اجرا هستند، بدین معنی است که نخ‌های فعال به‌صورت همزمان روی منابع سخت‌افزاری **مختلف** (یا واحدهای پردازنده مختلف) در حال اجرا هستند. اما چنانچه چندین نخ اجرایی به صورت **همروند** در حال اجرا باشند بدین معنی است که اجرای نخ‌ها روی **یک** منبع سخت‌افزاری زمان‌بندی شده است. در سیستم‌های همروند، نخ‌های فعال **آماده** اجرا هستند اما در هر لحظه از زمان تنها یک نخ توسط پردازنده در حال اجراست و سیستم‌عامل وظیفه گرفتن پردازنده از یک نخ اجرایی و تخصیص آن به یکی از نخ‌های فعال را دارد.

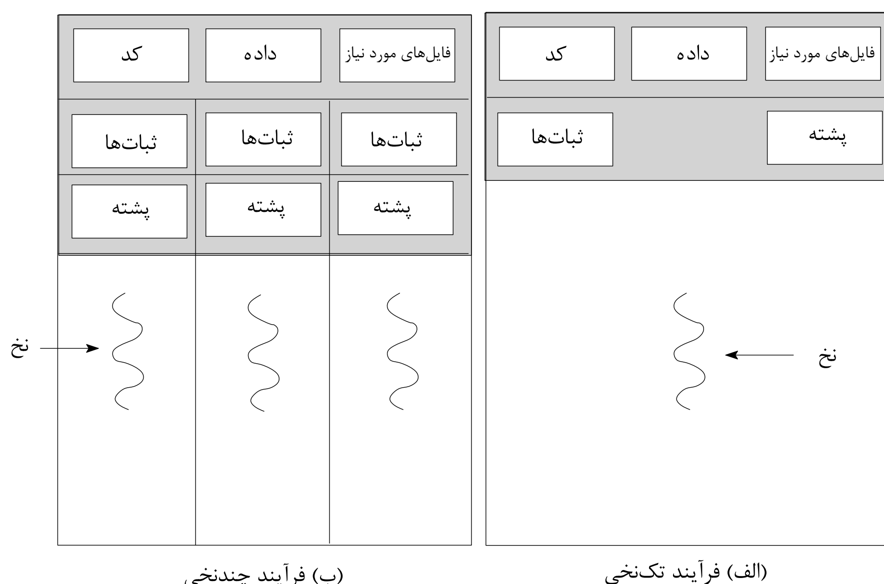
---

[۴۹] Thread

[۵۰] Single-threading:

[۵۱] Multithreading:

[۵۲] Concurrent



شکل ۱۱.۱: فرآیند تک نخی و فرآیند چندنخی

در تکنولوژی جدید **چندنخی همزمان**[۵۳] یا SMT، به چندین نخ اجازه اجرای همزمان روی یک پردازنده داده می‌شود. به عنوان مثال اگر یک نخ منتظر تکمیل عملیات ممیز شناور پردازنده باشد، همزمان نخ دیگری می‌تواند از واحد اعداد صحیح پردازنده استفاده کند اما دو نخ نمی‌توانند همزمان از یک واحد عملیاتی پردازنده مانند واحد محاسبات ممیز شناور استفاده کنند. در این تکنولوژی بخش‌هایی از پردازنده مانند واحدهای اجرایی، بین نخ‌ها به اشتراک گذاشته می‌شود در حالی که بخش‌های دیگر آن مانند ثبات‌های وضعیت و وقفه به تعداد نخ‌ها وجود دارد. بدون تکنولوژی SMT تنها یک نخ در هر زمان قابل اجراست. پیاده‌سازی اینتل از SMT به‌عنوان تکنولوژی مافوق نخ[۵۴] یا HT شناخته می‌شود.

همان‌طور که اشاره شد، مفهوم چندنخی امکان اجرای همروند برنامه‌ها را فراهم می‌کند اما به منظور این‌که موازی‌سازی به معنای واقعی اجرا گردد باید همروندی، همراه با چندین منبع سخت‌افزاری به‌کار گرفته شود. از این‌رو سازندگان قطعات سخت‌افزاری با به‌کارگیری جدیدترین پیشرفت‌های تکنولوژی، چندین هسته اجرایی را بر روی یک باریکه مدار مجتمع که **تراشه چندپردازنده**[۵۵] یا CMP نامیده می‌شود، قرار می‌دهند. ارتباط بین هسته‌ها در داخل یک تراشه به چند طریق پیاده‌سازی می‌گردد. به‌عنوان مثال یک هسته می‌توانند حافظه‌های

[۵۳] Simultaneous multithreading

[۵۴] Hyper-Threading

[۵۵] Chip multiprocessor

www.jozve.org

کش خود را با هسته‌های دیگر به اشتراک بگذارد و یا این‌که هر پردازنده حافظه کش مختص به خود را داشته باشد و یا ترکیبی از این دو (شکل ۱۳.۱). برخی از توپولوژی‌های شبکه‌ای رایج جهت اتصال هسته‌ها به یکدیگر، توپولوژی‌های حلقوی، مش، باس و کراس‌بار است. همچنین هسته‌ها ممکن است از روش‌های ارتباطی بین هسته‌ای مانند روش حافظه مشترک و یا انتقال پیام استفاده کنند.

شکل ۱۲.۱، سیر تکاملی پردازنده‌های چندهسته‌ای را نشان می‌دهد.

شکل ۱۲.۱: سیر تکامل پردازنده‌های چندهسته‌ای

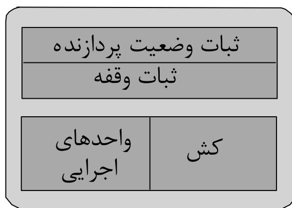
تکنولوژی HT در واقع ابزاری است که در آن برنامه‌نویس قادر است از منابع بیکار پردازنده به منظور انجام کار بیشتر استفاده کند و در نتیجه کارایی پردازنده تا میانگین ۳۰ درصد افزایش می‌یابد. این تکنولوژی وقتی با تکنولوژی چندهسته‌ای ترکیب شود، فرصت بهینه‌سازی بیشتری را فراهم می‌کند و در واقع عملکرد سیستم افزایش می‌یابد. شکل ۱۳.۱ تفاوت بین معماری‌های تک هسته‌ای، چندپردازنده‌ای و چندهسته‌ای را به طور ساده نشان می‌دهد.

مطالب اصلی این بخش، با استفاده از مراجع [۱، ۲۵] تنظیم شده است.

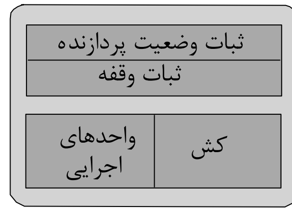
۲.۸.۱ برنامه‌نویسی سیستم‌های چندهسته‌ای

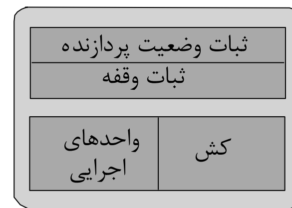
با توجه به آنچه ذکر گردید، یک سیستم چندهسته‌ای در واقع نوعی از سیستم‌های چندپردازنده‌ای است که همه پردازنده‌ها روی یک تراشه یکسان قرار می‌گیرند. پردازنده‌های چندهسته‌ای فضای کمتری را اشغال می‌کنند و با مصرف انرژی کمتر، کارایی بالاتری را ارئه می‌دهند از این‌رو برای استفاده در سیستم‌هایی که با باطری کار می‌کنند مناسب‌ترند. علاوه بر این، مجاورت چندین هسته پردازنده روی یک قطعه باعث می‌شود تا مدارهای الکترونیکی کنترل‌کننده سازگاری کش‌ها با نرخ ساعت بیشتری عمل کنند زیرا سیگنال‌های کنترلی در مسافت کمتری منتقل شده و در نتیجه نسبت به مدارهای کنترلی سیستم‌های چندپردازنده‌ای دیرتر تضعیف می‌گردند. همچنین سیستم‌های چندهسته‌ای زمان پاسخگویی سیستم را بهبود می‌بخشد.

برای این‌که از قابلیت‌های سیستم‌های چندهسته‌ای به درستی استفاده شود، لازم است که برنامه‌ها به صورت موازی نوشته شوند تا قابلیت اجرای موازی بر روی هسته‌های مختلف وجود داشته باشد. بنابراین اولین قدم در برنامه‌نویسی سیستم‌های چندهسته‌ای، طراحی الگوریتم‌های موازی کارا است. در طراحی الگوریتم‌های موازی



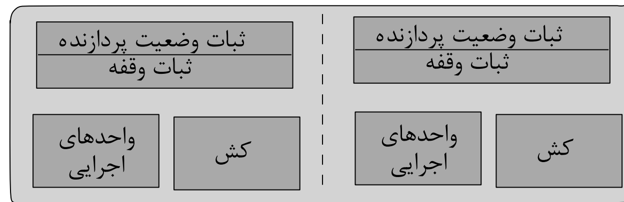
تک هسته‌ای



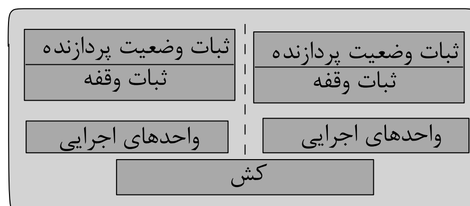


چند پردازنده‌ای

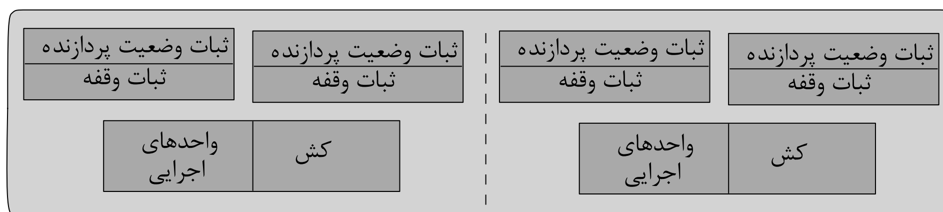
تکنولوژی مافوق نخ



چند هسته‌ای



چندهسته‌ای همراه با کش مشترک



ترکیب چندهسته‌ای با تکنولوژی فوق نخ

شکل ۱۳.۱: مقایسه ساده معماری‌های تک هسته‌ای، چندپردازنده‌ای و چندهسته‌ای

برای سیستم‌های چندهسته‌ای، لازم است که تمام مواردی که در بخش ۵.۱ به آن اشاره گردید، مورد توجه قرار گیرد.

www.jozve.org

فصل ۲

معرفی مدل برنامه‌نویسی مورد نیاز

www.jozve.org

همانطور که در بخش ۵.۱ اشاره گردید، لازم است که قبل از ارائه یک الگوریتم موازی مناسب، مدلی برای برنامه‌نویسی موازی انتخاب گردد و سپس الگوریتم در این مدل طراحی گردد. مدلی که برای مسئله محاسبه موازی غشای محدب مجموعه‌ای از نقاط، انتخاب شده است یکی از انواع مدل‌های انتقال پیام به‌نام مدل **چندکامپیوتری دانه‌درشت**[۱] (CGM) است. هدف از این فصل، معرفی این مدل و برخی مفاهیم مورد استفاده الگوریتم موازی محاسبه غشای محدب است که این الگوریتم، در فصل ۳ به تفصیل شرح داده خواهد شد. لازم به ذکر است که اکثر مطالب این فصل براساس مقالات دهنی[۲] و همکارانش [۱۵،۱۶] تنظیم شده است.

۱.۲ مفهوم دانه‌درشت و دانه‌ریز

برای اجرای الگوریتم‌های موازی در سیستم‌های مبادله پیام متشکل از p پردازنده، داده‌های ورودی به طور مساوی بین پردازنده‌ها تقسیم می‌شوند (تجزیه دامنه‌ای) و یا دستورات الگوریتم بین پردازنده‌ها به طور متعادل توزیع می‌گردند (تجزیه تابعی). برای آشنایی بیشتر با مفهوم تجزیه تابعی و دامنه‌ای بخش ۵.۱ را ببینید.
در تجزیه دامنه‌ای، هر پردازنده مسئول اجرای یک **پردازه**[۳] بر روی $\frac{n}{p}$ داده خواهد بود. یک پردازه (فرآیند)، اساساً یک برنامه در حال اجرا است و در صورت نیاز به مبادله اطلاعات بین پردازه‌ها، این ارتباط از طریق انتقال پیام بین پردازنده‌ها صورت می‌گیرد. در سیستم‌هایی که محدودیت خاصی بر روی تعداد پردازنده‌ها وجود ندارد و فرض می‌شود که تقریباً به تعداد n داده ورودی مسئله، پردازنده در اختیار هست، مقدار داده‌های تخصیص یافته به هر پردازنده از $\frac{n}{p} = O(۱)$ است. چنین سیستم‌هایی را اصطلاحاً **دانه‌ریز** می‌نامند. در مقابل، در سیستم‌هایی که تعداد پردازنده‌های موجود به مراتب کمتر از تعداد داده‌های ورودی است ($p << n$)، تعداد داده‌های تخصیص یافته به هر پردازنده به طور قابل ملاحظه‌ای بیشتر از $O(۱)$ است. در نتیجه: $O(۱) \ll O\left(\frac{n}{p}\right)$. چنین سیستم‌هایی را در اصطلاح **دانه‌درشت** می‌نامند.
در تجزیه تابعی، هر پردازنده مسئول اجرای لیستی از دستورات است که به آن محول شده است و این دستورات در قالب یک پردازه اجرا می‌گردند. در سیستم‌هایی که فرض می‌شود محدودیت خاصی بر روی تعداد پردازنده‌ها وجود ندارد، تعداد دستورالعمل‌هایی که به هر پردازنده تخصیص می‌یابد، کم است و در نتیجه برای اجرای

---

[۱] Coarse Grained Multicomputers

[۲] Dehne

[۳] Process

www.jozve.org

دستورات نیاز به ارتباطات بیشتری بین پردازنده‌ها است و اغلب زمان اجرای یک پردازه، صرف برقراری ارتباط با پردازنده‌های دیگر می‌شود. در مقابل در سیستم‌هایی که تعداد پردازنده‌ها نامحدود نیست، هر پردازه تعداد زیادی دستورالعمل متوالی را در اختیار دارد و زمان زیادی را صرف اجرای آن می‌نماید بنابراین نیاز کمتری به برقراری ارتباط با پردازنده‌های دیگر دارد و اغلب زمان اجرای پردازه صرف عملیات محاسباتی می‌گردد. با توجه به آنچه ذکر شد، اندازه یک پردازه در سیستم مبادله پیام به وسیله پارامتری به نام دانه‌بندی[۴] پردازه توصیف می‌شود. این پارامتر به صورت زیر بیان می‌گردد [۲۰]:

$$\text{دانه‌بندی پردازه} = \frac{\text{زمان انجام محاسبه}}{\text{زمان برقراری ارتباط}}$$

دو نوع دانه‌بندی وجود دارد که عبارتند از:

۱. **دانه‌بندی درشت**[۵]: هر پردازه تعداد زیادی دستورالعمل متوالی را در اختیار می‌گیرد و زمان زیادی صرف اجرای آن‌ها می‌نماید و نیاز کمتری به ارتباط با پردازنده‌های دیگر دارد. در نتیجه زمان برقراری ارتباطات نسبتاً کم است.

۲. **دانه‌بندی ریز**[۶]: هر پردازه حاوی تعداد کمی دستورالعمل متوالی است (حتی به اندازه یک دستورالعمل) و جهت اجرا نیاز به برقراری ارتباطات مکرر با پردازنده‌های دیگر دارد. در نتیجه زمان برقراری ارتباطات زیاد است.

مفاهیم دانه‌بندی درشت و دانه‌بندی ریز در تجزیه تابعی به ترتیب معادل مفاهیم دانه‌ریز و دانه‌درشت در تجزیه دامنه‌ای است.

۲.۲ مدل‌های محاسباتی دانه‌ریز

تا قبل از سال ۱۹۹۳ اغلب کارهای تئوری انجام شده در مسائل هندسه محاسباتی موازی، بر روی سیستم‌های دانه‌ریز متمرکز بودند و اغلب، هزینه‌های ارتباطات نادیده گرفته می‌شد که چنین فرضی در عمل غیرممکن

[۴] Granularity

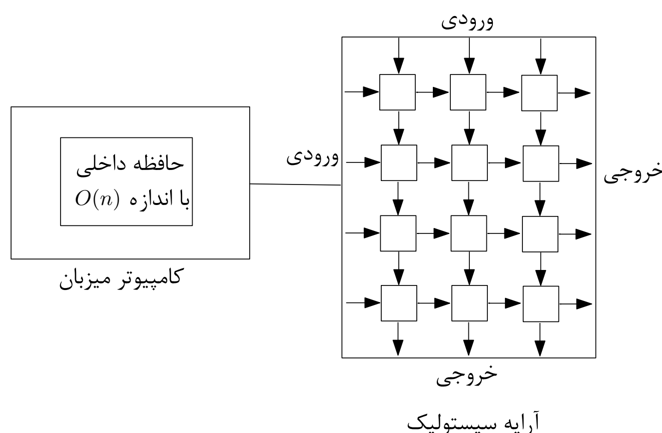
[۵] Coarse Granularity

[۶] Fine Granularity

است. بنابراین اغلب این الگوریتم‌ها جنبه تئوری داشته و در عمل کارایی چندانی نداشتند. درسال ۱۹۸۹ مدل دانه‌ریزی توسط اتلا[۷] و تسی[۸] [۴]، ارائه گردید که در آن، یک کامپیوتر میزبان با حافظه $O\left(n\right)$، به یک آرایه‌ی سیستولیک[۹] p-تایی با حافظه داخلی $O\left(1\right)$ برای هر واحد پردازش، متصل می‌شد (شکل ۱.۲).

یک **آرایه سیستولیک** شامل ردیف‌هایی از واحدهای پردازش داده[۱۰] است که سلول نامیده می‌شود. واحدهای پردازش داده شبیه واحد پردازنده مرکزی است با این تفاوت که فاقد ثبات شمارنده برنامه است. هر سلول داده‌های خود را بلافاصله پس از پردازش، با سلول‌های مجاور خود به اشتراک می‌گذارد. این آرایه‌ها اغلب مستطیل شکل هستند و داده‌ها در آن‌ها، همزمان بین پردازنده‌های مجاور حرکت می‌کنند.

در این مدل اکثر زمان محاسبات، صرف خواندن ورودی از حافظه کامپیوتر میزبان و بارگذاری آن در حافظه



شکل ۱.۲: مدل کامپیوتر ارائه شده توسط تسی و اتلا.

یک‌تایی پردازنده‌ها می‌شد. به‌عنوان مثال، شکل ۲.۲ نحوه اجرای الگوریتم مرتب‌سازی یک آرایه خطی را در این مدل نشان می‌دهد. آرایه ورودی شامل p کلید است که در هر واحد زمانی یک کلید توسط ماشین میزبان دریافت و در حافظه اولین سلول آرایه بارگذاری می‌شود. هر سلول مقدار دریافتی از سمت چپ را با مقدار ذخیره شده در ثبات محلی خود مقایسه می‌کند (همه ثبات‌ها در ابتدا مقدار $+\infty$ را ذخیره می‌کنند). مقدار کوچکتر در ثبات محلی نگهداری شده و مقدار بزرگتر به سمت راست شیفت داده می‌شود. وقتی که p کلید ورودی دریافت شد، $p-1$ چرخه ارتباطی اضافی لازم است تا تمام داده‌هایی که در حال انتقال هستند در

[۷] Atallah

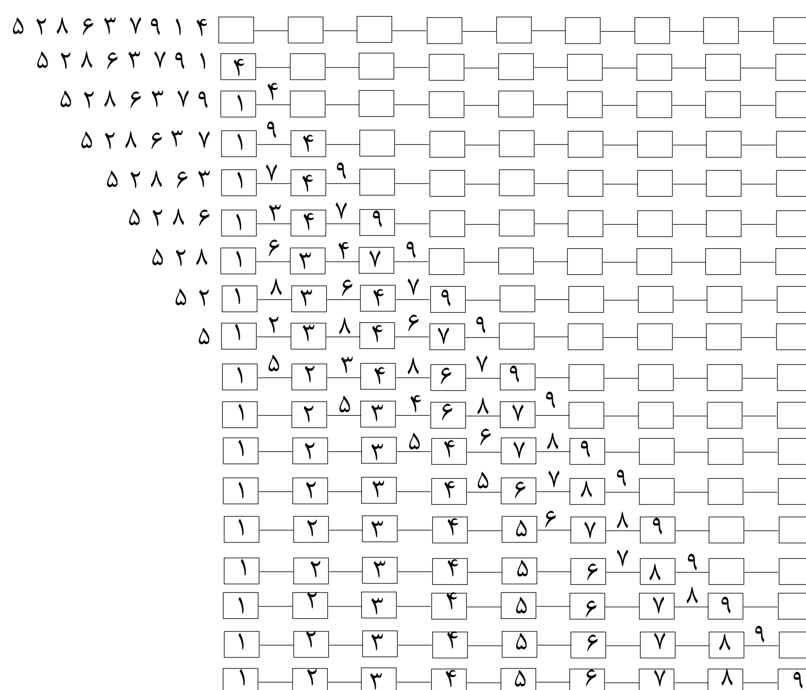
[۸] Tsay

[۹] Systolic Array

[۱۰] Data Processing Unit

www.jozve.org

مکان مناسب خود در آرایه سیستولیک قرار گیرند. اگر لیست مرتب شده نهایی از سمت چپ آرایه خارج شود، فاز خروجی می‌تواند بلافاصله پس از دریافت آخرین مقدار ورودی شروع شود. به این‌صورت که به محض ورود آخرین کلید، مقدار سلول اول با کلید ورودی مقایسه می‌شود و مقدار کوچک‌تر، به‌عنوان خروجی از سمت چپ خارج می‌شود. همزمان، جهت حرکت داده‌های در حال انتقال نیز، به سمت چپ تغییر می‌یابد. در نتیجه تنها از نصف سلول‌ها استفاده خواهد شد. به‌عنوان مثال در شکل ۲.۲ تا ردیف نهم، جهت انتقال داده‌ها به سمت راست است و از آنجا به بعد، جهت انتقال و مقایسه به سمت چپ خواهد بود. بنابراین زمانی‌که لیست نهایی از سمت چپ خارج می‌شود، مرتب‌سازی تنها با استفاده از یک آرایه سیستولیک به اندازه نصف داده‌های ورودی، قابل انجام است و زمان مرتب‌سازی صرفاً برابر با زمان ورودی- خروجی است [۳۰].



شکل ۲.۲: مرتب‌سازی یک آرایه خطی با استفاده از یک آرایه سیستولیک.

www.jozve.org

۳.۲ مدل‌های محاسباتی دانه‌درشت

برای کاستن از تنگناهای ورودی-خروجی[۱۱] مدل‌های محاسباتی دانه‌ریز و همچنین با الهام از مدل «حافظه‌های خارجی» که توسط گودریچ[۱۲] در سال ۱۹۹۵ معرفی شد [۲۲]، معماری کاملاً متفاوتی از سیستم‌های دانه‌ریز ارائه گردید که در آن هر کدام از پردازنده‌ها دارای حافظه قابل توجهی مختص به خود بودند. مجموع کل داده‌های مسئله در هر یک از حافظه‌های محلی سریعاً بارگذاری می‌شد و تا پایان حل مسئله در آنجا باقی می‌ماند که این بارگذاری اولیه با هدف کاهش تعداد عملیات ورودی- خروجی به پردازنده انجام می‌شد. اغلب الگوریتم‌های به‌کار رفته در سیستم‌های دانه‌ریز، مستقیماً قابل استفاده در این معماری جدید (معماری دانه‌درشت) نبودند و ترجمه این الگوریتم‌ها برای استفاده در معماری دانه‌درشت اغلب کار دشواری بود. بنابراین از سال ۱۹۹۵، حرکت به سمت مدل‌های محاسباتی دانه‌درشت [۱۲،۱۵،۲۴،۳۵] و به تبعیت از آن، الگوریتم‌های دانه‌درشت [۱۴، ۱۷، ۱۹، ۲۱، ۲۶، ۳۷] آغاز شد.

در ادامه به معرفی اجمالی مدل دانه‌درشت BSP [۱۳] پرداخته می‌شود و سپس مدل دانه‌درشت «**چندکامپیوتری دانه‌درشت**» (CGM) که مدل اصلی مورد بحث این پایان‌نامه است و از جهاتی مشابه مدل BSP است، معرفی می‌گردد.

۱.۳.۲ مدل BSP

مدل BSP، یکی از مدل‌های محاسباتی دانه‌درشت است که در سال ۱۹۹۰، توسط والیانت[۱۴] معرفی گردید [۳۵]. یک کامپیوتر BSP مجموعه‌ای از ماژول‌های حافظه و پردازنده‌هاست که پردازنده‌ها به‌وسیله یک مسیریاب[۱۵] که پیام‌ها را به روش نقطه به نقطه هدایت می‌کند، به یکدیگر متصل شده‌اند. محاسبات مدل BSP، به توالی از ابرگام‌ها[۱۶] که با موانع همگام‌سازی[۱۷] از یکدیگر تفکیک می‌شوند، تقسیم‌بندی می‌شوند. یک ابرگام متشکل از

---

[۱۱] Input-Output

[۱۲] Goodrich

[۱۳] Bulk Synchronous Parallel

[۱۴] Valiant

[۱۵] Router

[۱۶] Supersteps

[۱۷] Barrier

www.jozve.org

گام‌های محاسباتی و گام‌های ارتباطی است. پارامترهای تاثیرگذار در این مدل عبارتند از:

n: تعداد داده‌های ورودی

p: تعداد پردازنده‌ها

L: کمترین زمان بین گام‌های همزمانی که بر حسب زمان یک عمل محاسباتی اصلی اندازه‌گیری می‌شود.

g: نسبت سرعت محاسبات محلی به پهنای باند مسیریاب. سرعت محاسبات محلی برابر تعداد عملیات محاسباتی در واحد زمان و پهنای باند مسیریاب برابر تعداد پیام‌های با اندازه واحد است که می‌تواند در واحد زمان توسط مسیریاب، مسیریابی شود.

در هر ابرگام، به هر پردازنده شرکت‌کننده در ابرگام یک وظیفه که شامل گام‌های محاسباتی محلی و گام‌های ارتباطی است، محول می‌گردد. در یک گام محاسباتی، هر پردازنده محاسبات مورد نیاز را بر روی داده‌های حافظه محلی خود، مستقل از پردازنده‌های دیگر انجام داده و سپس در گام ارتباطی، چنانچه به ارتباط با پردازنده دیگری نیاز داشته باشد، به مسیریاب درخواست داده و ارتباط نقطه به نقطه انجام می‌شود. بعد از هر دوره زمانی L، یک بررسی سراسری انجام می‌شود تا مشخص گردد که آیا تمام پردازنده‌ها وظایف خود را تکمیل نموده‌اند یا نه. اگر چنین باشد ماشین ابرگام بعدی را آغاز می‌کند وگرنه دوره زمانی بعدی به این ابرگام ناتمام، تخصیص می‌یابد.

اگر λ تعداد ابرگام‌ها، ω^{i}_{Comp} زمان محاسبات ابرگام i-ام و t_j تعداد عملیات محاسباتی اصلی‌ای باشد که توسط پردازنده j-ام در ابرگام i-ام انجام می‌شود، آنگاه زمان کل محاسبات انجام شده در این مدل یاT_{Comp} برابر با رابطه زیر است:

$$T_{Comp} = \sum_{i=1}^{\lambda} \omega^{i}_{Comp} \qquad \omega^{i}_{Comp} = max\{L, t_1, \dots, t_j, \dots, t_p\} \qquad (۲-۱)$$

همچنین اگر λ تعداد ابرگام‌ها و $\omega^{i}_{Comm,j}$ هزینه برقراری ارتباط با پردازنده j-ام در ابرگام i-ام باشد، آنگاه زمان کل ارتباطات مورد نیاز در این مدل یاT_{Comm} برابر با رابطه زیر است:

$$T_{Comm} = \sum_{i=1}^{\lambda} \omega^{i}_{Comm} \qquad \omega^{i}_{Comm} = max_{j=1}^{p}\{\omega^{i}_{Comm,j}\} \qquad (۲-۲)$$

فرض کنید پردازنده p_j در طول ابرگام i-ام، پیام‌هایی به طول $r_1, \dots, r_{j'}$ را دریافت و پیام‌هایی به طول $s_1, \dots, s_{j''}$ را ارسال کند. در این صورت:

www.jozve.org

$$\omega^{i}_{Comm,j} = max\left\{L, g\left(\sum_{u=۱}^{j'} r_u + \sum_{u=۱}^{j''} s_u\right)\right\} \quad (۳-۲)$$

مطالب این بخش با استفاده از مراجع [۱۶، ۳۵] تنظیم شده است.

۲.۳.۲ مدل CGM

اغلب **چندکامپیوترهای**[۱۸] بزرگ، شامل مجموعه‌ای از p پردازنده با جدیدترین تکنولوژی هستند که هر یک حافظه محلی قابل ملاحظه‌ای مختص به خود دارند و با استفاده از شبکه‌های داخلی به‌هم متصل می‌شوند. این ماشین‌ها را در اصطلاح دانه‌درشت می‌نامند به این معنی که تعداد داده‌های تخصیص یافته به پردازنده‌ها به طور قابل ملاحظه‌ای، بزرگتر از $O\left(۱\right)$ است.

مدل چندکامپیوتری دانه‌درشت که به طور مختصر با $CGM\left(n,p\right)$ نشان داده می‌شود، در سال ۱۹۹۳ توسط دهنی[۱۹] و همکارانش ارائه گردید [۱۵]. این مدل تنها دارای دو پارامتر تاثیرگذار n و p است و تشکیل شده است از:

- تعداد p پردازنده که یک مسئله با اندازه ورودی n را حل می‌کنند. این پردازنده‌ها به ترتیب با $p_۱, \ldots, p_p$ شماره‌گذاری می‌شوند و هر پردازنده تنها شماره i مربوط به خود را می‌شناسد اما از موقعیت فیزیکی خود در ماشین موازی اطلاعی ندارد.

- هر پردازنده p_i، یک حافظه محلی مختص به خود، به اندازه $O\left(\frac{n}{p}\right)$ دارد و طبق تعریف دانه‌درشت $O\left(\frac{n}{p}\right) \gg O\left(۱\right)$ است، به عبارت دیگر تعداد داده‌های تخصیص یافته به هر پردازنده یا اندازه حافظه محلی هر پردازنده به طور قابل ملاحظه‌ای بیشتر از $O\left(۱\right)$ است.

- پردازنده‌ها از طریق شبکه‌های ارتباطی دلخواه و یا از طریق حافظه مشترک به یکدیگر متصل می‌شوند. چنانچه پردازنده‌ها از طریق حافظه مشترک با یکدیگر ارتباط برقرار کنند، در این‌صورت می‌بایست فضایی در حافظه جهت تبادل اطلاعات در نظر گرفته شود. انواع شبکه‌های ارتباطی مورد استفاده در مدل CGM، شامل فوق مکعبی مانند Intel iPSC/860، شبکه درختی مانند CM-5 و مش دو بعدی مانند Intel Paragon است. برای توضیحات بیشتر در مورد انواع شبکه‌های ارتباطی بخش ۳.۱ را

[۱۸] Multicomputers

[۱۹] Dehne

www.jozve.org

ببینید. از میان شبکه‌های موجود، شبکه ارتباطی مش و انواع آن برتری دارند، زیرا بسیاری از مسائل واقعی، دارای داده‌هایی هستند که به‌طور طبیعی قابل نگاشت بر روی شبکه مش هستند. هر پردازنده ممکن است پیام‌های $\log n$ بیتی را با هر یک از پردازنده‌های مجاور خود در زمان ثابت مبادله کند و با هر پردازنده غیرمجاور خود از طریق انواع عملیات ارتباطی اصلی، تبادل پیام داشته باشد.

- سکوی محاسباتی به‌کار رفته در این مدل، طبق طبقه‌بندی فلین از دسته SIMD می‌باشد. بخش ۲.۱ را ببینید. در این دسته از طبقه‌بندی، تمامی پردازنده‌ها برنامه یکسانی را اما روی داده‌های متفاوت اجرا می‌کنند که این داده‌ها در حافظه محلی هر یک از پردازنده‌ها ذخیره شده‌اند.

تعریف ارائه شده از مدل CGM، برگرفته از منابع [۱۵، ۳۲، ۳۶] است.

۴.۲ طرح کلی یک الگوریتم در مدل CGM

یک الگوریتم در مدل CGM شامل توالی از دورهای محاسباتی و دورهای ارتباطی است که توسط یک مانع همگام‌سازی از هم جدا می‌شوند. این مانع می‌تواند یک پیام کنترلی یا یک وقفه‌ای باشد که از طرف سیستم‌عامل به تمامی پردازنده‌ها ارسال می‌گردد. طرح کلی الگوریتم در مدل CGM با الگوریتم ۳ نشان داده می‌شود که در آن R تعداد دورهای کلی الگوریتم است.

الگوریتم ۳ : طرح کلی الگوریتم CGM

برای مقادیر $i = ۱$ تا $i = R$ اجرا کن:

{

i- امین دور محاسباتی

i- امین دور ارتباطی

}

یک دور محاسباتی معادل ابرگام محاسباتی در مدل BSP است و هزینه کل محاسبات T_{Comp} به صورت مشابه تعریف می‌شود. برای توضیحات بیشتر درباره مدل BSP، بخش ۱.۳.۲ را ببینید. در هر دور محاسباتی، به منظور پردازش اطلاعات محلی هر پردازنده، معمولاً از بهترین الگوریتم ترتیبی ممکن استفاده می‌شود.

www.jozve.org

یک دور ارتباطی شامل یک رابطه h-تایی[20] است که در آن هر پردازنده حداکثر h، $h \leq \frac{n}{p}$ داده خود را با پردازنده‌های دیگر مبادله می‌کند. هزینه ω^i_{Comm} هر دور ارتباطی مقدار یکسانی دارد که با $H_{n,p}$ نشان داده می‌شود. بنابراین هزینه کل ارتباطات یک الگوریتم CGM با λ دور ارتباطی برابر $T_{Comm} = \lambda H_{n,p}$ است.

مزیت اصلی مدل CGM در برابر BSP:

مزیت اصلی مدل CGM و علت برتری آن نسبت به مدل BSP، این است که اجازه می‌دهد هزینه ارتباطات یک الگوریتم موازی، تنها با یک پارامتر منفرد، که تعداد دورهای ارتباطی λ است، سنجیده شود. یک الگوریتم CGM در واقع یک الگوریتم BSP است که در گام ارتباطی آن، به جای این‌که برای مبادله پیام‌های کوچک بین هر دو پردازنده، یک سری سربارهای داده‌ای نظیر ایجاد کانال‌های ارتباطی، تنظیم پروتکل‌های ارتباط و غیره هزینه گردد، کلیه ارتباطات بین پردازنده‌ها همزمان و توسط یک رابطه h-تایی انجام می‌شود. توضیحات بیشتر در خصوص نحوه اجرای این عمل در بخش ۵.۲ ارائه می‌گردد.

هر چند که λ پارامتر اصلی تعیین کننده در الگوریتم‌های مدل CGM است، اما برای ارزیابی آن‌ها پارامترهای دیگری نیز در نظر گرفته می‌شود. به طور کلی، سه فاکتور مهم برای ارزیابی الگوریتم‌های دانه‌درشت وجود دارد:

۱. مقدار محاسبات محلی مورد نیاز

۲. تعداد و نوع دورهای ارتباطی مورد نیاز

۳. مقیاس‌پذیری[21]

فاکتور **مقیاس‌پذیری** نشان‌دهنده میزان بهبود کارایی یک سیستم موازی، در اثر افزایش منابع محاسباتی است یا به عبارت دیگر منعکس‌کننده توانایی سیستم، در به کارگیری منابع پردازشگر بیشتر است. بنابراین چنانچه ضریب تسریع یک سیستم موازی متناسب با افزایش تعداد پردازنده‌ها، افزایش یابد، سیستم مقیاس‌پذیر خواهد بود. در مدل CGM، با افزایش تعداد پردازنده‌ها، تعداد داده‌های موجود در حافظه هر یک از پردازنده‌ها یا $\frac{n}{p}$، کاهش می‌یابد. بنابراین در این مدل، الگوریتمی مقیاس‌پذیر خواهد بود که برای دامنه وسیعی از مقادیر $\frac{n}{p}$ کارا و قابل اجرا باشد.

---

[20] h-relation

[21] Scalability

www.jozve.org

زمان اجرای یک الگوریتم در مدل CGM برابر با مجموع کل زمان محاسبات محلی یعنی T_{Comp} و کل زمان سپری شده برای ارتباطات بین پردازنده‌ها یعنی T_{Comm} است. بنابراین طراحان الگوریتم، می‌بایست سعی در نوشتن الگوریتم‌هایی نمایند که علاوه بر کاهش زمان محاسبات، تعداد دورهای ارتباطی را نیز به حداقل برساند. از آنجایی‌که هزینه هر دور ارتباطی برابر $H_{n,p}$ است بنابراین در مدل CGM برای محاسبه هزینه ارتباطات تنها، تعداد دورهای ارتباطی سنجیده می‌شود.

مطالب این بخش با استفاده از مراجع [۳۲،۳۶] تنظیم شده است.

۵.۲ انواع عملیات ارتباطی در مدل CGM

مطالب اصلی این بخش با استفاده از مراجع [۱۵، ۳۰،۳۶] تنظیم شده است.

۱.۵.۲ مفاهیم کلی ارتباطات

در شبکه‌های ارتباطی، کوچکترین واحد منطقی انتقال داده **بسته**[۲۲] نامیده می‌شود. بسته می‌تواند حاوی یک پیام و یا بخشی از یک پیام بزرگتر باشد. یک پیام به صورت مجموعه‌ای از اطلاعات مرتبط که با همدیگر به‌عنوان یک موجودیت حرکت می‌کنند، در نظر گرفته می‌شود. پیام می‌تواند یک دستورالعمل، داده، یک سیگنال همزمانی و یا یک سیگنال وقفه باشد. به‌عنوان مثال یک رکورد اطلاعاتی که از چند فیلد داده‌ای تشکیل شده است، می‌تواند به عنوان یک پیام تلقی گردد. پیام‌های بزرگ، به k بسته تقسیم شده و در هر واحد زمانی یک بسته ارسال می‌گردد. به هر یک از این بسته‌ها یک شماره ترتیبی تخصیص داده می‌شود که نشان دهنده اجزای پیام است (در اغلب موارد واژه بسته و پیام معادل هم و در یک معنا به کار برده می‌شوند).

هر بسته یا پیام به طور معمول دارای یک **سرآیند**[۲۳] است که آدرس مقصد و همچنین یک‌سری اطلاعات مسیریابی که توسط فرستنده پیام، به پیام اضافه می‌گردد، را نگه‌داری می‌کند. بدنه پیام که بارمفید[۲۴] پیام نامیده می‌شود، داده‌های واقعی را حمل می‌کند. اطلاعات مسیریابی واقع در سرآیند، بسته به نوع الگوریتم مسیریابی که استفاده می‌شود، ممکن است توسط گره‌های میانی تغییر یابد. در اکثر معماری‌های پردازنده‌ها،

[۲۲] Packet

[۲۳] Header

[۲۴] Payload

www.jozve.org

هر مبادله پیام بین دو پردازنده، نیازمند یک سری سربارهای داده‌ای نظیر ایجاد کانال‌های ارتباطی، تنظیم پروتکل‌های ارتباط و غیره می‌باشد که مستقل از اندازه پیام است. همچنین هر پیام ممکن است شامل اطلاعات دیگری نظیر تشخیص خطا و یا اطلاعات تصحیح باشد که برای سادگی از این اطلاعات و سربارهای داده‌ای صرف‌نظر می‌شود.

در زمینه محاسبات دانه‌درشت، جنبه‌های ارتباطی مورد نیاز سیستم‌های موازی، به طور معمول از سیستم انتقال پیام الگو گرفته است. در اکثر سیستم‌های موازی برای پیاده‌سازی عملیات ارتباطی مورد نیاز الگوریتم‌ها، از یک‌سری کتابخانه‌های استاندارد عرضه شده در بازار، استفاده می‌شود. استفاده از این کتابخانه‌های استاندارد سبب می‌شود تا برنامه‌ها قابلیت حمل و انعطاف بر روی سکوهای مختلف سخت‌افزاری و معماری‌های مختلف شبکه را داشته باشند. در این صورت از منظر برنامه‌نویسی، عمل انتقال پیام تنها به فراخوانی یک زیرروال از داخل کد برنامه خلاصه می‌شود که این زیرروال‌ها در داخل کتابخانه تعبیه شده است. از میان کتابخانه‌ها و واسط‌های ارتباطی رایج، می‌توان به PVM، CMMD و کتابخانه Intel iPSC/860، کتابخانه IBM SP2 و MPI اشاره نمود. در اغلب کتابخانه‌های موجود، عملیات ارتباطی به دو دسته کلی زیر تقسیم‌بندی می‌شوند [۳۰]:

۱. ارتباطات **نقطه به نقطه**[۲۵]: وقتی یک پردازنده، پیامی را مستقل از پردازنده‌های دیگر، تنها به یک پردازنده ارسال می‌کند، یک عمل ارتباطی نقطه به نقطه انجام شده است.

۲. ارتباطات **جمعی**[۲۶] : در عملیات جمعی یک یا چند پردازنده پیام یکسان یا متفاوتی را به یک یا چندین پردازنده دیگر ارسال می‌کند. برخی از عملیات ارتباط جمعی عبارتند از: انتشار[۲۷]، پراکندن[۲۸]، گردآوری[۲۹] و ترکیب[۳۰].

ماهیت هر یک از این عملیات ارتباطی در مدل‌های مختلف انتقال پیام، یکسان بوده و تنها در نحوه پیاده‌سازی متفاوت می‌باشند. بنابراین در ادامه، به معرفی انواع عملیات ارتباطی مدل مورد بحث این فصل و نحوه پیاده‌سازی

---

[۲۵] Point-to-point

[۲۶] Collective

[۲۷] Broadcast

[۲۸] Scatter

[۲۹] Gather

[۳۰] Combine

www.jozve.org

هر یک از این عملیات پرداخته می‌شود.

در مدل CGM، در هر دور ارتباطی، نیاز به تبادل اطلاعات بین پردازنده‌ها است تا هر پردازنده با استفاده از اطلاعات دریافتی، دور محاسباتی بعدی را آغاز نماید. همه ارتباطات سراسری در این مدل، با مجموعه کوچکی از عملیات ارتباطی استاندارد که اغلب در سطح سخت افزار قابل اجراست مانند: انتشار بسته[۳۱]، تجمیع بسته [۳۲]، انتشار کلی[۳۳]، تبادل کلی[۳۴]، جمع جزئی[۳۵](اسکن) و مرتب‌سازی کلی[۳۶]؛ پیاده‌سازی می‌گردد. عمل ارتباطی اصلی معرفی شده در این مدل، مرتب‌سازی کلی است. درباره سایر عملیات ارتباطی مذکور، چنانچه توسط سخت افزار پشتیبانی نشوند، می‌توان آن‌ها را بر حسب یک تعداد ثابتی از عملیات مرتب‌سازی کلی پیاده‌سازی نمود.

۲.۵.۲ عمل ارتباطی اصلی: مرتب‌سازی کلی

همان‌طور که قبلاً تعریف شد، در یک رابطه h-تایی، هر پردازنده، h داده خود را با پردازنده‌های دیگر مبادله می‌کند. هر پردازنده p_i دو آرایه OUT_i و IN_i دارد که هر یک $O\left(h\right)$ قلم داده[۳۷] دارد و به ترتیب میانگیر خروجی و میانگیر ورودی نامیده می‌شود. منظور از قلم داده کوچکترین واحد اطلاعاتی در یک سیستم است، به عنوان مثال هر عنصر آرایه را، یک قلم داده می‌نامند. هر پردازنده p_i، میانگیر خروجی خود را به p زیرآرایه $OUT_{i,۱},\dots,OUT_{i,p}$ تقسیم‌بندی می‌کند. یک رابطه h-تایی شامل مسیردهی $OUT_{i,j}$، $۱\leq i,j\leq p$ از پردازنده p_i به پردازنده p_j است. هر پردازنده p_j زیرآرایه‌های $OUT_{i,j}$، $۱\leq i\leq p$ را دریافت می‌کند و آن را در میانگیر ورودی خود IN_j ذخیره می‌کند. به عنوان مثال شکل ۳.۲ قسمت (الف)، میانگیر خروجی سه پردازنده مختلف را نشان می‌دهد. چنانچه به هر یک از عناصر زیرآرایه‌ها شماره پردازنده مقصد (اندیس دوم زیرآرایه)، به عنوان برچسب تخصیص داده شود و سپس تمامی عناصر تمام زیرآرایه‌ها، بر حسب این برچسب

---

[۳۱] Segment broadcast

[۳۲] Segment gather

[۳۳] All-to-All broadcast

[۳۴] Total exchange

[۳۵] Partial sum

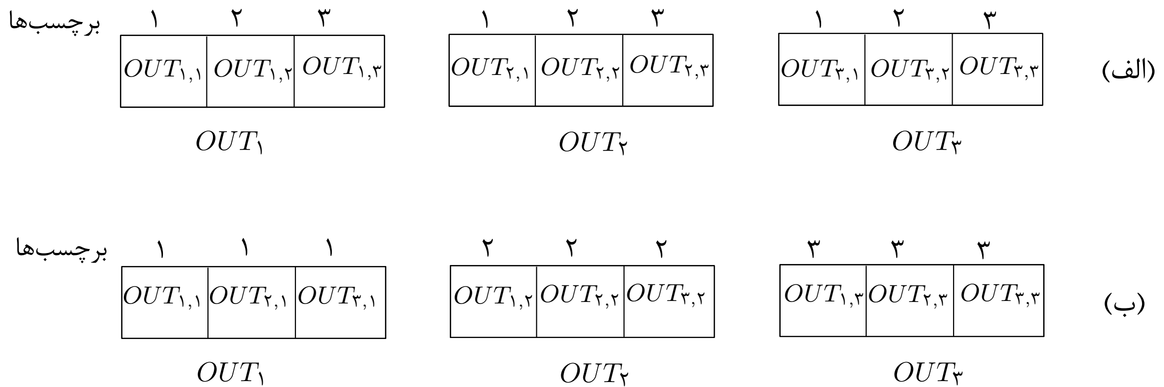
[۳۶] Global sort

[۳۷] Data item

www.jozve.org

مرتب شوند، در نتیجه این مرتب‌سازی، عناصر به نحوی جابه‌جا می‌گردند که در نهایت تمام عناصر با برچسب i به میانگیر پردازنده p_i منتقل می‌شوند. شکل (ب) نتیجه عمل مرتب‌سازی بر روی عناصر زیرآرایه‌های شکل (الف) را نشان می‌دهد که در واقع همان نتیجه رابطه h-تایی است که باید در IN_j ذخیره گردد.

بنابراین در واقع عمل مبادله پیام‌هایی به طول h بین پردازنده‌ها را می‌توان با یک عمل مرتب‌سازی کلی که



شکل ۳.۲: پیاده‌سازی یک رابطه h-تایی.

به صورت زیر تعریف می‌شود پیاده‌سازی نمود.

تعریف ۱.۵.۲ در مدل $CGM\,(n,p)$، در حافظه محلی هر یک از پردازنده‌ها، $O\left(\frac{n}{p}\right)$ قلم داده ذخیره شده است که به هر قلم داده یک کلید نسبت داده می‌شود. کلید می‌تواند مقدار داده، شماره پردازنده و غیره باشد. عمل مرتب‌سازی $O\,(n)$ قلم داده‌ی ذخیره شده بر روی $CGM\,(n,p)$، برحسب کلید تعریفی، **مرتب‌سازی کلی** نامیده می‌شود.

پیچیدگی زمانی این عمل با $T_s\,(n,p)$ نشان داده می‌شود که بسته به نوع شبکه ارتباطی و الگوریتم‌های مسیریابی به‌کار رفته در هر یک از این شبکه‌ها، متفاوت است. از آنجایی‌که بهترین الگوریتم ترتیبی مرتب‌سازی از مرتبه $\Theta\,(n\log n)$ است، بنابراین چنانچه $T_s\,(n,p)$ برابر با $O\left(\frac{n\log n}{p}\right)$ باشد، الگوریتم مرتب‌سازی کلی بهینه خواهد بود.

به عنوان مثال در شبکه ارتباطی مش، $T_s\,(n,p)$ برابر با $\Theta\left(\frac{n}{p}\left(\log n+\sqrt{p}\right)\right)$ است [۶] که این پیچیدگی در صورت برقراری شرط زیر بهینه خواهد بود:

$$T_s\,(n,p)=O\left(\frac{n\log n}{p}\right)\quad\longrightarrow\quad\frac{n}{p}\log n\geq\frac{n}{p}\sqrt{p}$$

www.jozve.org

$$\longrightarrow \quad \log n \geq \sqrt{p}$$

$$\longrightarrow \quad n \geq ۲^{\sqrt{p}}$$

همچنین در شبکه ارتباطی فوق مکعبی، $T_s\left(n,p\right)$ برابر با $\Theta\left(\frac{n}{p}\left(\log n + \log^۲ p\right)\right)$ است [۲۷] که این پیچیدگی در صورت برقراری شرط زیر بهینه خواهد بود:

$$T_s\left(n,p\right) = O\left(\frac{n\log n}{p}\right) \quad \longrightarrow \quad \frac{n}{p}\log n \geq \frac{n}{p}\log^۲ p$$

$$\longrightarrow \quad \log n \geq \log^۲ p$$

$$\longrightarrow \quad n \geq ۲^{(\log p)(\log p)}$$

$$\longrightarrow \quad n \geq p^{\log p}$$

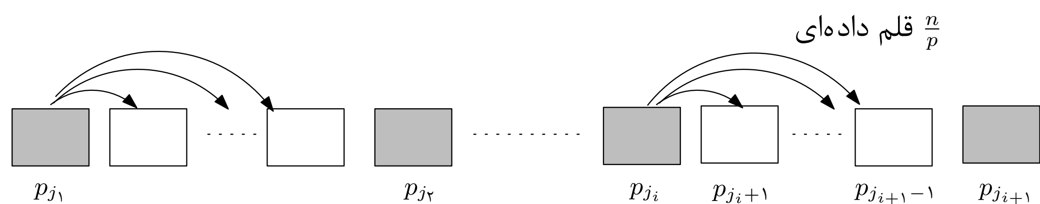
۳.۵.۲ سایر عملیات ارتباطی برپایه مرتب‌سازی کلی

در ادامه پنج عمل ارتباطی دیگر در مدل CGM، تعریف می‌گردند. تمامی این عملیات‌ها، همان‌طور که در لم۷.۵.۲ نشان‌داده خواهد شد، با تعداد ثابتی از عملیات مرتب‌سازی کلی و محاسبات محلی از مرتبه $O\left(\frac{n}{p}\right)$ پیاده‌سازی می‌گردند. البته در برخی از شبکه‌های ارتباطی، پیاده‌سازی مستقیم این عملیات با استفاده از الگوریتم‌های مسیریابی شبکه ([۳۰]، فصل‌های ۱۰ و ۱۴)، در ثابت‌های پیچیدگی، بهتر از پیاده‌سازی با عمل مرتب‌سازی کلی خواهد بود.

تعریف ۲.۵.۲ عمل انتشار بسته: در این عمل $p \leq q$ پردازنده با شماره‌های $j_۱ < j_۲ < \cdots < j_q$ انتخاب می‌شوند. هر پردازنده p_{j_i}، لیستی از $۱ \leq k \leq \frac{n}{p}$ داده محلی خود را به پردازنده‌های $p_{j_i+۱}, \ldots, p_{j_{i+۱}-۱}$ ارسال می‌کند.

شکل۴.۲ را ببینید. در این شکل، پردازنده‌های خاکستری رنگ، پردازنده‌های منتخب جهت انتشار داده هستند. پیچیدگی زمانی این عمل با $T_{sb}\left(n,p\right)$ نشان داده می‌شود.

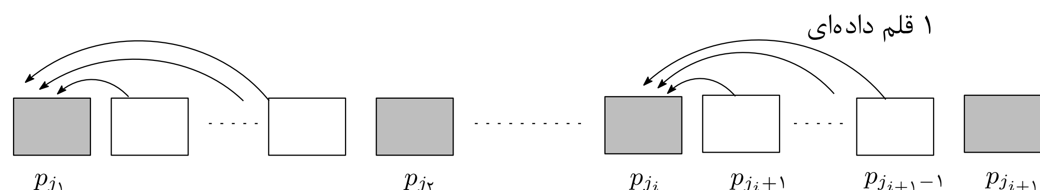
www.jozve.org



شکل ۴.۲: عمل انتشار بسته

تعریف ۳.۵.۲ عمل تجمیع بسته: در این عمل $q \le p$ پردازنده با شماره‌های $j_1 < j_2 < \cdots < j_q$ انتخاب می‌شوند. هر پردازنده p_{j_i}، یک داده از پردازنده‌های $p_{j_i+1}, \ldots, p_{j_{i+1}-1}$ دریافت می‌کند.

شکل ۵.۲ را ببینید. در این شکل، پردازنده‌های خاکستری رنگ، پردازنده‌های منتخب جهت دریافت داده هستند. این عمل، عکس عمل انتشار بسته است و قبل از هر چیز، باید مطمئن بود که پردازنده‌های انتخابی، برای دریافت تمام پیام‌های ارسال شده، به اندازه کافی حافظه دارند. پیچیدگی زمانی این عمل با $T_{sg}\left(n,p\right)$ نشان داده می‌شود.

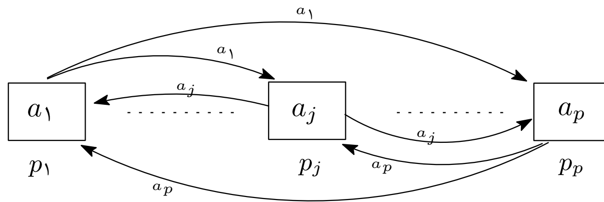


شکل ۵.۲: عمل تجمیع بسته

تعریف ۴.۵.۲ عمل انتشار کلی: در این عمل، هر پردازنده یک پیام مشابه را، به تمام پردازنده‌های دیگر ارسال می‌کند. شکل ۶.۲ را ببینید، در این شکل هر پردازنده p_j، داده متعلق به خود، a_j، را به تمامی پردازنده‌های دیگر ارسال می‌کند. پیچیدگی زمانی این عمل با $T_b\left(p\right)$ نشان داده می‌شود.

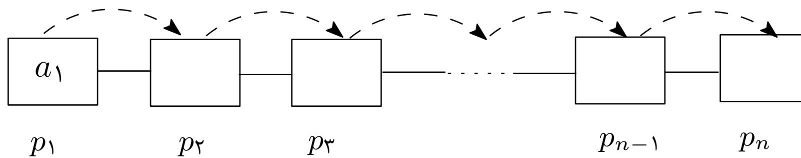
پیچیدگی این عمل، چنانچه در سطح سخت‌افزار و مطابق با الگوریتم‌های مسیریابی مختص هرشبکه ارتباطی، پیاده‌سازی گردد برابر با قطر شبکه خواهد بود. قطر هر شبکه ارتباطی برابر با بیشترین فاصله بین هر جفت پردازنده است. به عنوان مثال در یک شبکه خطی، همانطور که در شکل ۷.۲ مشاهده می‌شود، به منظور انتشار داده پردازنده p_1، پردازنده p_1 داده خود را به پردازنده سمت راست خود ارسال می‌کند. این پردازنده یک کپی

www.jozve.org



شکل ۶.۲: عمل انتشار کلی

از داده دریافتی ایجاد کرده و داده را به پردازنده سمت راست خود انتقال می‌دهد و به همین ترتیب. بنابراین زمان انتشار برابر با $O\left(p\right)$ است.

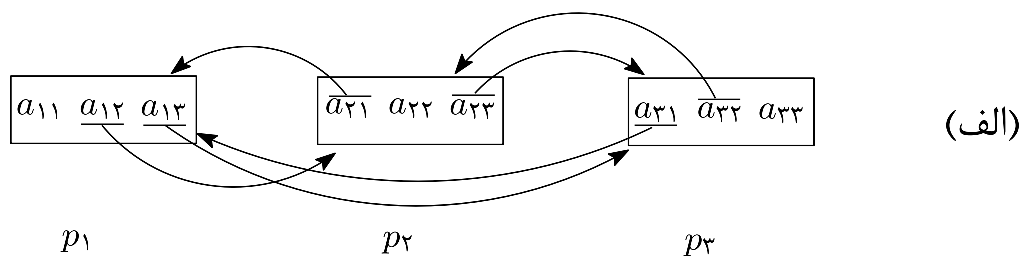


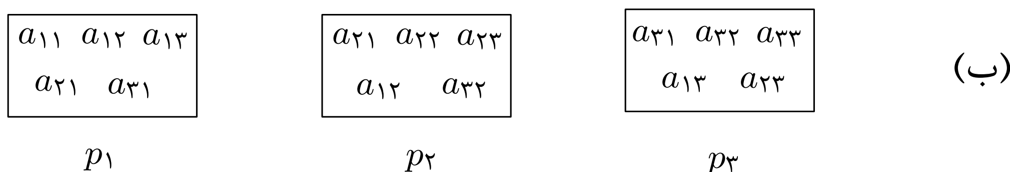
شکل ۷.۲: پیاده‌سازی عمل انتشار کلی در یک آرایه خطی در سطح سخت‌افزار.

تعریف ۵.۵.۲ عمل تبادل کلی: در این عمل، هر پردازنده پیام متفاوتی را به سایر پردازنده‌ها ارسال می‌کند. به عنوان مثال فرض کنید هر پردازنده در ابتدا، p قلم داده را در حافظه خود ذخیره دارد. در طی عمل تبادل، هر پردازنده اولین داده خود را به پردازنده p_1، دومین داده را به پردازنده p_2 و به همین ترتیب ارسال می‌کند. پس از انجام این عمل، هر پردازنده، p داده (با احتساب داده متعلق به خود)، از هر پردازنده یک داده، دریافت کرده است. بنابراین هر پردازنده p_i، i-امین داده پردازنده‌های دیگر را دریافت می‌کند.
شکل ۸.۲ را ببینید، شکل (الف) محتوای حافظه هر پردازنده را، قبل از عمل تبادل کلی و شکل (ب) محتوای حافظه را، پس از عمل تبادل نشان می‌دهد. برای اجرای این عمل، لازم است که هر یک از پردازنده‌ها فضای خالی‌ای حداقل به اندازه p در اختیار داشته باشند. پیچیدگی زمانی این عمل با $T_x\left(p\right)$ نشان داده می‌شود.

تعریف ۶.۵.۲ عمل جمع جزئی(اسکن): در این عمل p داده، از هر پردازنده یک داده، با یکدیگر ترکیب شده و یک نتیجه حاصل می‌گردد. این نتیجه به تمام پردازنده‌ها ارسال می‌گردد. عمل ترکیب می‌تواند هر یک از عمل‌های شرکت‌پذیر ریاضی یا منطقی مانند جمع، ماکزیمم، ضرب و نظایر آن باشد. پیچیدگی زمانی این عمل با $T_p\left(p\right)$ نشان داده می‌شود.

www.jozve.org





شکل ۸.۲: عمل تبادل کلی

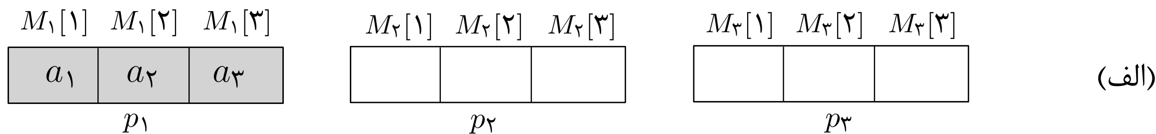
در ادامه روش کلی انجام عمل انتشار بسته با یک مثال ساده روشن می‌شود. فرض کنید سه پردازنده p_1، p_2 و p_3 در اختیار است و لازم باشد که سه قلم داده‌ی موجود در حافظه پردازنده اول، عیناً به پردازنده‌های بعدی منتقل شود. اگر در حافظه محلی هر یک از پردازنده‌های p_2 و p_3 سه قلم خالی ایجاد شود (در شکل ۹.۲ قسمت الف، $M_i\left[j\right]$ نشان‌دهنده قلم داده ذخیره شده در خانه j حافظه پردازنده i-ام است) و سپس این قلم‌ها به حافظه محلی پردازنده اول منتقل شوند و به نحوی مرتب گردند که ابتدا قلم‌های اول هر یک از پردازنده‌ها به ترتیب شماره پردازنده در کنار هم، سپس قلم‌های دوم پردازنده‌ها و در نهایت قلم‌های سوم پردازنده‌ها در کنار هم قرار گیرند (شکل ۹.۲ قسمت ب)، در این صورت به راحتی می‌توان محتویات پردازنده اول را در قلم‌های خالی پردازنده‌های دیگر کپی کرد (شکل ۹.۲ قسمت ج). بعد از عمل کپی چنانچه قلم‌ها بر حسب شماره پردازنده مجدداً مرتب گردند، هر قلم به پردازنده خود باز می‌گردد با این تفاوت که این‌بار با داده مورد نظر پر شده است (شکل ۹.۲ قسمت د).

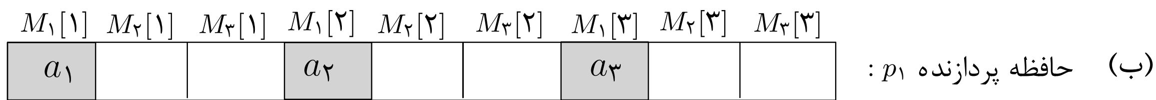
عمل انتشار بسته همان‌طور که در لم ۷.۵.۲ رابطه (۱) نشان داده خواهد شد، با $O\left(1\right)$ عمل مرتب‌سازی کلی و محاسبات محلی از مرتبه $O\left(\frac{n}{p}\right)$ قابل پیاده‌سازی است. برای اثبات این رابطه، ابتدا عمل **«انتشار بسته یک‌تایی»** [۳۸] تعریف می‌شود. سپس نحوه شبیه‌سازی این عمل با عمل مرتب‌سازی کلی و برخی محاسبات محلی، نشان داده می‌شود و در نهایت مشخص می‌شود که عمل انتشار بسته، قابل کاهش به عمل انتشار بسته یک‌تایی است.

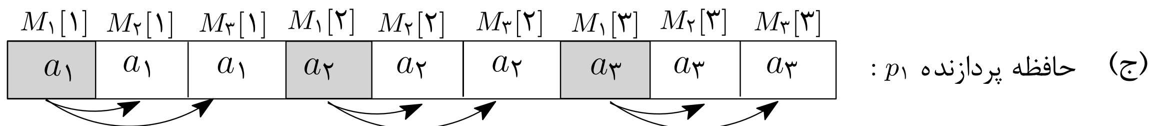
---

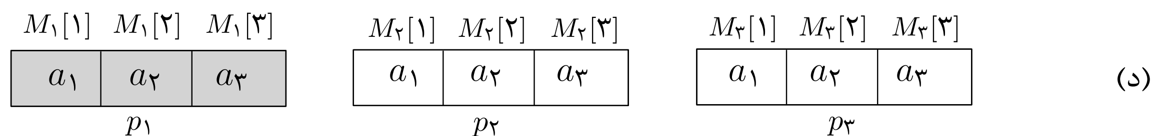
[۳۸] Segment 1-broadcast

www.jozve.org









شکل ۹.۲: بیان روش کلی انجام عمل انتشار بسته با یک مثال ساده.

عمل انتشار بسته یک‌تایی: تعداد $r \leq p$ پردازنده با شماره‌های $k_1 < k_2 < \cdots < k_r$ انتخاب می‌شوند. هر پردازنده p_{k_i}، تنها یک قلم داده را، به پردازنده‌های $p_{k_i}, \ldots, p_{k_{i+1}-1}$ ارسال می‌کند و هر یک از این پردازنده‌ها، از داده دریافتی، $O\left(\frac{n}{p}\right)$ کپی، در حافظه محلی خود ایجاد می‌کنند. شکل ۱۰.۲ قسمت (الف) را ببینید.

لم ۷.۵.۲ ([۱۵]) برای هر $CGM\left(n,p\right)$ با $\frac{n}{p} \geq p$ روابط زیر برقرار است:

۱. $T_{sb}\left(n,p\right) = O\left(\frac{n}{p} + T_s\left(n,p\right)\right)$

۲. $T_{sg}\left(n,p\right) = O\left(\frac{n}{p} + T_s\left(n,p\right)\right)$

۳. $T_b\left(p\right) = O\left(\frac{n}{p} + T_s\left(n,p\right)\right)$

۴. $T_x\left(p\right) = O\left(\frac{n}{p} + T_s\left(n,p\right)\right)$

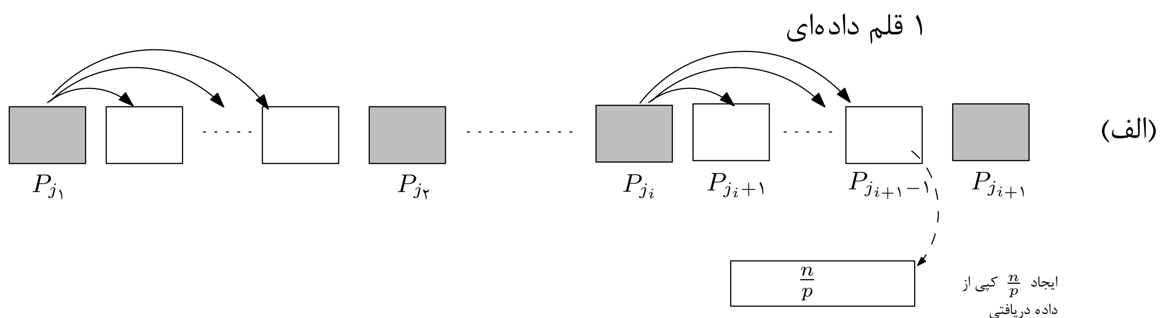
۵. $T_p\left(p\right) = O\left(\frac{n}{p} + T_s\left(n,p\right)\right)$

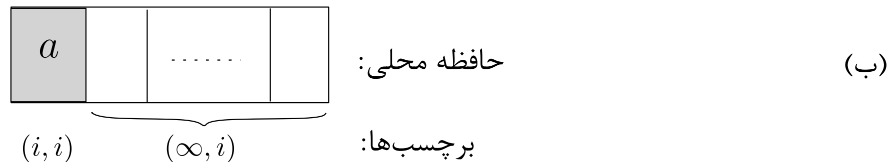
اثبات. **اثبات رابطه (۱):** عمل انتشار بسته یک‌تایی به روش زیر با تعداد ثابتی از عملیات مرتب‌سازی کلی و محاسبات محلی از مرتبه $O\left(\frac{n}{p}\right)$ قابل پیاده‌سازی است:

www.jozve.org

در ابتدا برای هر پردازنده، $\frac{n}{p}$ قلم داده ایجاد می‌شود. این قلم‌ها، یا داده‌هایی هستند که می‌بایست منتشر شوند و یا قلم‌های خالی هستند. هر یک از قلم‌ها، به روش زیر برچسب‌گذاری می‌شوند (شکل ۱۰.۲ قسمت ب):

- در هر پردازنده‌ای که می‌بایست داده خود را منتشر کند، به داده مورد نظر، برچسب (i, i) و به سایر قلم‌های خالی آن، برچسب (∞, i) داده می‌شود. i شماره پردازنده و ∞ یک عدد خیلی بزرگ است.
- در هر پردازنده‌ای که هیچ داده‌ای را منتشر نمی‌کند، به **اولین** قلم حافظه برچسب (i, i) و به سایر قلم‌های خالی، برچسب (∞, i) تخصیص داده می‌شود.





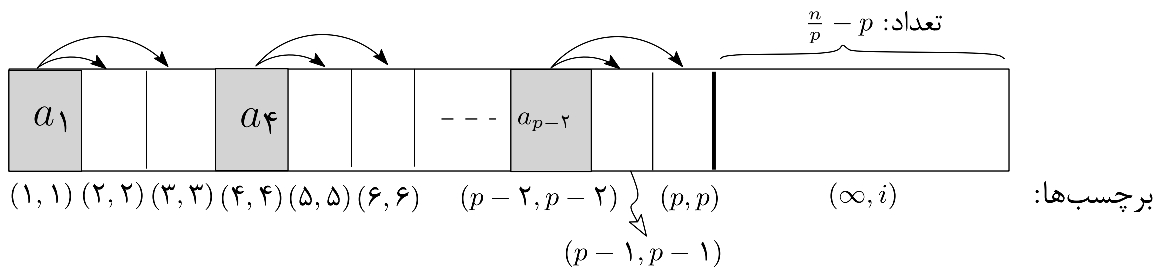
شکل ۱۰.۲: عمل انتشار بسته یک‌تایی و نحوه برچسب‌گذاری.

سپس با استفاده از یک مرتب‌سازی کلی و با در نظر گرفتن اولین مولفه برچسب، به عنوان کلید مرتب‌سازی، کلیه n قلم داده‌ای واقع در پردازنده‌ها، مرتب می‌شوند. در نتیجه، اولین داده از هر پردازنده، به ترتیب شماره پردازنده، در p خانه اول حافظه محلی اولین پردازنده، قرار می‌گیرند و هر قلم داده‌ای پر، در قلم‌های خالی بعدی، کپی می‌شود (شکل ۱۱.۲). این عمل کپی به صورت محلی و در زمان $O(p) \leq \left(\frac{n}{p}\right)$ انجام می‌شود.

پس از عمل کپی، کلیه قلم‌های داده، یک‌بار دیگر و این‌بار برحسب مولفه دوم برچسب (شماره پردازنده)، مرتب می‌شوند که سبب می‌شود هر یک، به حافظه محلی پردازنده خود، بازگردانده شوند. در نهایت در هر پردازنده، قلم پر به صورت محلی در سایر خانه‌های حافظه کپی می‌شود.

در واقع عمل انتشار بسته یک‌تایی،به منظور انجام عمل کپی، با یک عمل مرتب‌سازی، داده‌ها را در یک حافظه، گردآوری می‌کند و پس از انجام عمل کپی به صورت محلی، با یک عمل مرتب‌سازی دیگر، داده‌ها را به محل

www.jozve.org



شکل ۱۱.۲: عمل کپی در حافظه محلی پردازنده p_1

اصلی خود باز می‌گرداند.

عمل انتشار بسته، به‌صورت زیر قابل کاهش به عمل انتشار یک‌تایی است:

تعداد $q \leq p$ پردازنده $p_{j_1} < p_{j_2} < \cdots < p_{j_q}$ به‌منظور انتشار بسته انتخاب می‌شوند. عدد k به عنوان برچسب پردازنده p_i تعریف می‌شود اگر و تنها اگر $j_k \leq i < j_{k+1}$. تعداد $\frac{n}{p}$ قلم داده برای هر پردازنده ایجاد می‌شود که یا قلم‌های خالی است و یا قلم‌هایی است که می‌بایست انتشار یابند. تعداد کل قلم‌های داده، n می‌باشد که باید به صورت کلی مرتب شوند. به‌این صورت که هر قلم داده‌ای x مربوط به پردازنده p_i $(1 \leq i \leq p)$ به ترتیب کلیدهای زیر مرتب می‌شوند:

۱. کلید اول: برچسب p_i

۲. کلید دوم: رتبه x در لیست محلی پردازنده p_i که حداکثر این لیست $\frac{n}{p}$ عنصر دارد

۳. کلید سوم: شماره پردازنده (i)

بعد از این مرتب‌سازی، لیست کلی n عنصری از همه قلم‌های روی همه پردازنده‌ها در نظر گرفته می‌شود. چنانچه y قلمی از این لیست باشد که می‌بایست منتشر شود، این قلم در ق خالی بعد از آن، قلم‌های خالی بعدی که مولفه اولشان یکسان است، کپی می‌گردد. هر قلم y و خانه‌های خالی مجاور آن اغلب داخل یک پردازنده قرار می‌گیرند و بنابراین عمل کپی به صورت محلی قابل اجرا است. در مواردی که این قلم‌ها داخل یک پردازنده قرار نگرفته باشند، آخرین قلم داده‌ی پردازنده i-ام، طبق تعریف مدل ارائه شده در بخش ۲.۳.۲، از طریق شبکه ارتباطی و در زمان ثابت، قابل کپی به اولین قلم داده پردازنده $(i+1)$-ام است. فرآیند کپی قلم y در خانه‌های مجاور آن، قابل کاهش به عمل انتشار یک‌تایی است. بنابراین هر یک از پردازنده‌ها، عمل کپی را در زمان $O\left(\frac{n}{p}\right)$ انجام می‌دهند.

بعد از اتمام عمل کپی، لیست کلی یک‌بار دیگر اما با ترتیبی معکوس، مرتب می‌شود. یعنی هر قلم داده x،

www.jozve.org

ابتدا برحسب شماره پردازنده، سپس برحسب رتبه x و در نهایت برحسب برچسب p_i، مرتب می‌شود. با این مرتب‌سازی، تمامی داده‌های پردازنده‌ها، در جایگاه اولیه خود قرار می‌گیرند با این تفاوت که این‌بار، هر قلم داده با مقدار مورد نظر پر شده است و لیست حاصل، همان نتیجه عمل انتشار بسته خواهد بود. بنابراین عمل انتشار بسته با $O(۱)$ عمل مرتب‌سازی کلی در زمان $T_s(n,p)$ و محاسبات محلی از مرتبه $O\left(\frac{n}{p}\right)$ شبیه‌سازی می‌گردد.

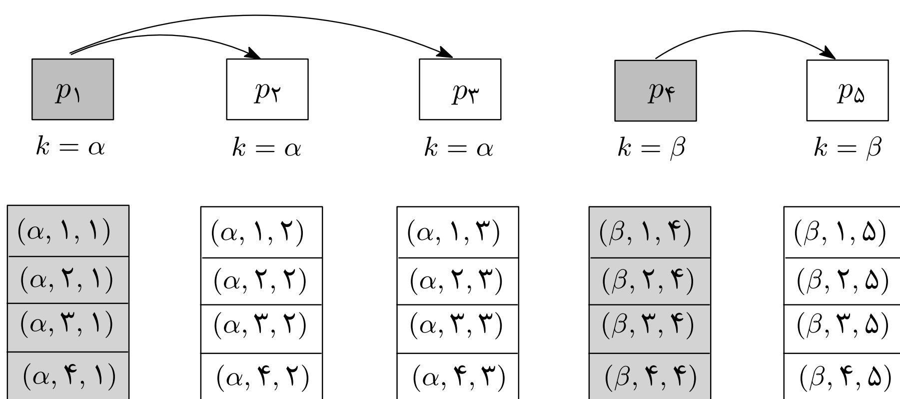
مثال ۸.۵.۲ شکل ۱۲.۲، عمل انتشار بسته را در $CGM\,(۲۰,۵)$ نشان می‌دهد. پردازنده‌های $p_۱$ و $p_۴$ جهت انتشار داده انتخاب می‌شوند. پردازنده $p_۱$ هر ۴ قلم داده‌ی درون حافظه محلی خود را به پردازنده‌های $p_۲$ و $p_۳$ و پردازنده $p_۴$ داده‌های خود را به پردازنده $p_۵$ ارسال می‌کند. به سه پردازنده اول برچسب α و به دو پردازنده آخر برچسب β تخصیص داده می‌شود. به هر یک از قلم‌های داده‌ای پردازنده‌ها، یک کلید سه‌تایی تعلق می‌گیرد. قسمت (الف) را ببینید. نتایج مرتب‌سازی کلی و عمل کپی بر روی هر پنج لیست، در قسمت (ب) نشان داده شده است. تمام قلم‌های داده‌ای با برچسب یکسان α، در ابتدای لیست قرار می‌گیرند. سپس قلم‌های اول هر یک از این پردازنده‌ها، به ترتیب شماره پردازنده کنار هم قرار می‌گیرند و با یک عمل کپی محلی، قلم داده پر، در هر یک از قلم‌های خالی مجاور آن، کپی می‌شود. علامت || نمایشگر انتقال سخت‌افزاری داده، بین دو پردازنده مجاور است.

هر یک از رابطه‌های ۲، ۳، ۴ و ۵، به طور مشابه با $O(۱)$ عمل مرتب‌سازی کلی و محاسبات محلی از مرتبه $O\left(\frac{n}{p}\right)$، قابل شبیه سازی است. در ادامه به منظور اثبات هر یک از این روابط، نحوه برچسب‌گذاری و همچنین ترتیب مرتب‌سازی در هر یک، به اختصار بیان می‌شود.

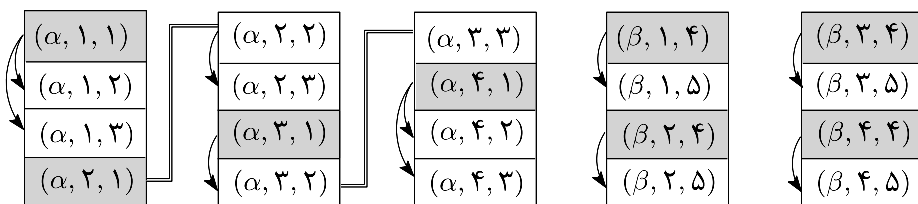
اثبات رابطه (۲):

درعمل تجمیع بسته، برای هر پردازنده p_i که قرار است از k پردازنده بعدی خود، داده دریافت کند، k قلم خالی در نظر گرفته می‌شود. این k قلم خالی را، قلم‌های رزرو شده نامند. به هر یک از قلم‌های رزرو شده، به ترتیب، برچسب $(i+l,i)$، $۱ \leq l \leq k$ و به سایر قلم‌های خالی برچسب (∞,i)، داده می‌شود. در هر پردازنده p_i که قرار است یک داده را انتشار دهد، به داده انتشار یافته برچسب (i,i) و به سایر خانه‌های خالی حافظه، یک قلم خالی با برچسب (∞,i) تخصیص داده می‌شود. پس از این برچسب‌گذاری، ابتدا داده‌ها بر حسب مولفه اول برچسب، مرتب می‌شوند. سپس هر قلم پر، در قلم خالی ماقبل کپی می‌شود و در نهایت داده‌ها بر حسب

www.jozve.org



(الف)



(ب)

شکل ۱۲.۲: عمل انتشار بسته در $CGM\ (۲۰, ۵)$

مولفه دوم برچسب مرتب می‌شوند.

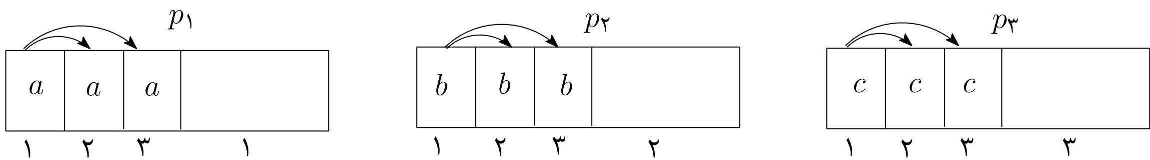
اثبات رابطه (۳):

در عمل انتشار کلی، در هر پردازنده p_i، از داده مورد نظر جهت انتشار، $p - ۱$ کپی ایجاد شده و به هر یک از داده‌ها (با احتساب داده اولیه)، برچسب ۱ تا p، تخصیص داده می‌شود. برای سایر خانه‌های خالی حافظه، یک قلم خالی با برچسب i در نظر گرفته می‌شود. مرتب‌سازی کلی با در نظر گرفتن برچسب (شماره پردازنده)، به عنوان کلید مرتب‌سازی اجرا می‌گردد. لیست حاصل، همان نتیجه عمل انتشار کلی است. شکل۱۳.۲ ، پیاده‌سازی عمل انتشار کلی را در $CGM\ (۹, ۳)$، نشان می‌دهد.

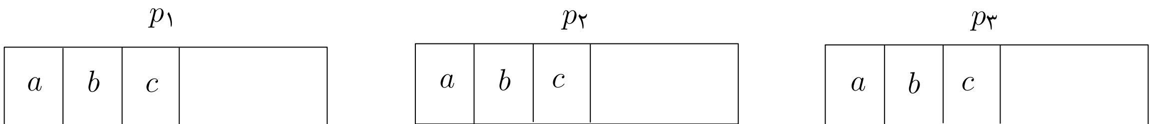
اثبات رابطه (۴):

در عمل تبادل کلی، هر پردازنده p_i، در ابتدا شامل p داده است و به هر یک از این داده‌ها برچسب i تخصیص داده می‌شود. هر یک از پردازنده‌ها، باید i-امین ($۱ \leq i \leq p$) داده خود را به پردازنده i-ام ارسال کند. برای پیاده‌سازی این عمل، هر پردازنده p_j، در حافظه محلی خود، از داده i-ام، $۱ \leq i \leq p$ و $i \neq j$ ، یک کپی با

www.jozve.org



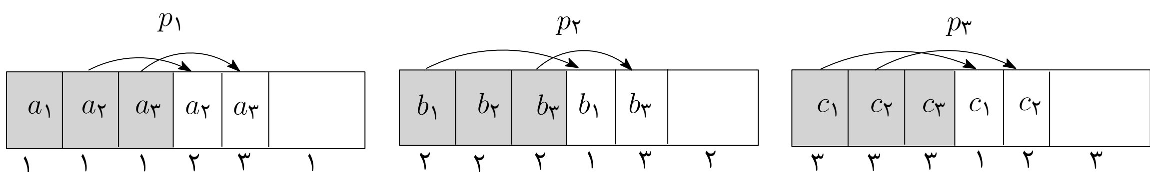
(الف) عمل کپی و برچسب‌گذاری داده‌های حافظه



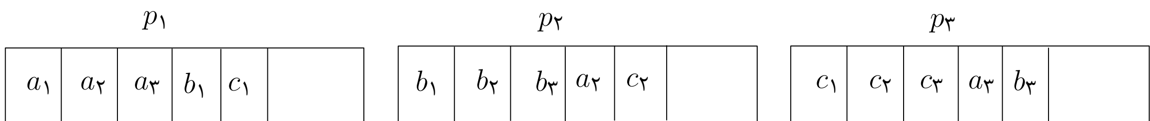
(ب) محتوای حافظه‌ها بعد از مرتب‌سازی کلی

شکل ۱۳.۲: عمل انتشار کلی در $CGM\ (۹,۳)$

برچسب i ایجاد می‌کند. برای سایر خانه‌های خالی حافظه، یک قلم خالی با برچسب j در نظر گرفته می‌شود. مرتب‌سازی کلی با در نظر گرفتن برچسب (شماره پردازنده)، به عنوان کلید مرتب‌سازی اجرا می‌گردد. لیست حاصل، همان نتیجه عمل تبادل کلی است. شکل۱۴.۲، پیاده‌سازی عمل تبادل کلی را در $CGM\ (۹,۳)$، نشان می‌دهد. در این شکل، قلم‌های خاکستری رنگ، داده‌هایی هستند که باید منتشر شوند.



(الف) عمل کپی و برچسب‌گذاری داده‌های حافظه



(ب) محتوای حافظه‌ها بعد از مرتب‌سازی کلی

شکل ۱۴.۲: عمل تبادل کلی در $CGM\ (۹,۳)$

اثبات رابطه (۵):

در عمل جمع جزئی، در هر پردازنده p_i، به قلم داده‌ای، برچسب $(i, rank, i)$ و به سایر خانه‌های خالی، قلم خالی با برچسب $(\infty, rank, i)$ تخصیص داده می‌شود. عدد $rank$ رتبه x در لیست محلی پردازنده p_i است.

www.jozve.org

مولفه‌های اول، دوم و سوم برچسب به ترتیب کلیدهای اول، دوم و سوم مرتب‌سازی کلی هستند. پس از انجام عمل مرتب‌سازی، کلیه داده‌هایی که می‌بایست با یکدیگر ترکیب شوند، در حافظه محلی اولین پردازنده قرار می‌گیرند. عمل ترکیب در زمان اسکن داده‌ها که برابر با $O\left(\frac{n}{p}\right) \leq O\left(p\right)$ است، انجام می‌شود و نتیجه حاصل در p قلم خالی بعدی کپی می‌گردد. این p قلم خالی، یا در حافظه محلی پردازنده اول و یا در صورتی‌که $\frac{n}{p} < ۲p$ باشد، در حافظه‌های محلی پردازنده‌های اول و دوم قرار دارد. بنابراین عمل کپی یا به صورت محلی و یا از طریق انتقال سخت‌افزاری بین دو پردازنده مجاور، در زمان $O\left(\frac{n}{p}\right)$ انجام می‌شود. در نهایت برای بازگرداندن هر قلم داده به جایگاه اولیه خود، لازم است که یک عمل مرتب‌سازی‌کلی با ترتیب معکوس بر روی لیست کلی داده‌ها، انجام شود. $\square$

یکی دیگر از عملیات پرکاربرد در یک ماشین CGM، عمل گروه‌بندی است. در این عمل، پردازنده‌های ماشین CGM در گروه‌هایی که با شناسه گروه مشخص می‌شوند، دسته‌بندی می‌گردند. این عمل به‌صورت یک رویه از کتابخانه CGM قابل دسترس است و با دستور $partitionCGM(intgroupId)$ فراخوانی می‌شود. عمل گروه‌بندی، یک عمل جمعی است به این معنا که تمام پردازنده‌هایی که همزمان این رویه را فراخوانی می‌کنند، عضو گروه شماره $groupId$ می‌شوند. در این صورت چنانچه پردازنده p_i عضو گروه G_k، به‌عنوان مثال عمل انتشار کلی را فراخوانی کند، پیام تنها به تمام پردازنده‌های گروه G_k ارسال می‌گردد [۱۰].

www.jozve.org

فصل ۳

الگوریتم موازی محاسبه غشای محدب

www.jozve.org

هدف از این فصل، بررسی یک الگوریتم موازی، قطعی و مقیاس‌پذیر در مدل چندکامپیوتری دانه‌درشت برای حل مسئله محاسبه غشای محدب یک مجموعه از نقاط در فضای دوبعدی است. این مسئله که در بخش ۱.۱.۳ معرفی می‌گردد، یکی از اولین مسائل هندسه محاسباتی است که برای آن الگوریتم موازی طراحی شده است. به عنوان مثال، در مدل موازی دانه‌ریز (رجوع کنید به بخش ۱.۲)، برای تعداد زیادی از معماری‌ها نظیر CRCW PRAM [۲]، CREW PRAM [۱۱]، فوق مکعبی [۲۹] و مش [۲۸]، الگوریتم محاسبه غشای محدب ارائه شده است. در مدل دانه‌درشت نیز، از سال ۱۹۹۳ الگوریتم‌هایی نظیر [۳۷،۱۵،۱۴] برای حل این مسئله ارائه شده است.

لازم به ذکر است که الگوریتم موازی مورد بحث این فصل، توسط دیالو[1] و همکارانش ارائه شده است و مطالب این فصل براساس مقاله [۱۹] تنظیم شده است. زمان اجرای این الگوریتم برابر $O\left(\frac{n\log n}{p} + T_s\left(n,p\right)\right)$ است و تنها به تعداد ثابتی از دورهای ارتباطی کلی و فضای حافظه $\frac{n}{p} \geq p^{\epsilon}$ احتیاج دارد که در آن ϵ یک مقدار ثابت بزرگتر از صفر است. همچنین این الگوریتم، بسیار مقیاس‌پذیر است به طوری‌که برای دامنه وسیعی از مقادیر $\frac{n}{p}$ کارا و قابل اجراست. از آنجایی که طبق رابطه ۳-۲، $T_{sequential}$، زمان محاسبه بهترین الگوریتم ترتیبی غشای محدب برابر با $\Theta\left(n\log n\right)$ است، الگوریتم مورد بحث، یا در زمان بهینه $\Theta\left(\frac{n\log n}{p}\right)$ و یا در زمان مرتب‌سازی $T_s\left(n,p\right)$ اجرا خواهد شد. در نتیجه زمان اجرای الگوریتم، زمانی که $\frac{T_{sequential}}{p}$ بیشتر از $T_s\left(n,p\right)$ باشد یا زمانی‌که $T_s\left(n,p\right)$ بهینه باشد، بهینه می‌شود.

۱.۳ محاسبه غشای محدب

یکی از اساسی‌ترین و مهم‌ترین مسائل هندسه محاسباتی مسئله محاسبه غشای محدب یک مجموعه از نقاط است. هندسه محاسباتی یکی از شاخه‌های علوم کامپیوتر است که برای حل مسائل هندسی از روش‌های الگوریتمی استفاده می‌کند و غشای محدب، یک ساختار هندسی پرکاربرد است که اغلب از آن به عنوان یک ساختار ابتدایی در حل مسائل پیشرفته هندسه محاسباتی نظیر پردازش تصویر، تشخیص الگو و سیستم‌های اطلاعاتی جغرافیایی[2]، محاسبه مثلث‌بندی مجموعه‌ای از نقاط و غیره استفاده می‌شود.

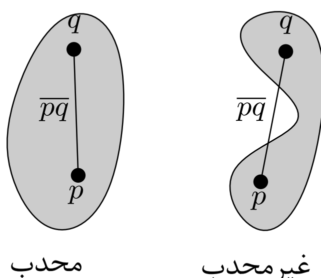
[1] Diallo

[2] Geographic Information System (GIS)

www.jozve.org

۳.۱.۱ غشای محدب یک مجموعه از نقاط در صفحه

تعریف ۱.۱.۳ یک زیرمجموعه S از صفحه **محدب**[۳] نامیده می‌شود اگر و تنها اگر برای هر جفت نقاط $p, q \in S$، پاره‌خط $\overline{pq}$ کاملاً در داخل S واقع شود. شکل۱.۳ را ببینید.



شکل ۱.۳: مقایسه دو مجموعه محدب و غیرمحدب

تعریف ۲.۱.۳ غشای محدب[۴] یک مجموعه S، کوچک‌ترین مجموعه محدبی است که S را در بر می‌گیرد و با $CH\left(S\right)$ نشان داده می‌شود. به عبارت دقیق‌تر، غشای محدب در واقع اشتراک تمام مجموعه‌های محدبی است که S را در بر می‌گیرد.

برای درک بهتر غشای محدب یک مجموعه متناهی P، شامل n نقطه در صفحه، می‌توان تصور نمود که نقاط میخ‌هایی هستند که در صفحه کوبیده شده‌اند و غشای محدب P، منحنی ایجاد شده توسط کش پلاستیکی است که دور میخ‌ها انداخته شده است. شکل ۲.۳ قسمت (آ) را ببینید. این دیدگاه منجر به تعریف دیگری از غشای محدب P می‌شود: **چندضلعی محدب یکتایی** که راس‌هایش نقاطی از P هستند و سایر نقاط P را در بر می‌گیرند. شکل ۲.۳ قسمت (آ) را ببینید.

تعریف ۳.۱.۳ غشای فوقانی[۵] بخشی از غشای محدب است که در حرکت ساعت‌گرد از سمت چپ‌ترین نقطه p_1 به سمت راست‌ترین نقطه p_k، پیمایش می‌شود. شکل ۲.۳ قسمت (ب) را ببینید. در حرکت از سمت راست‌ترین نقطه به سمت چپ‌ترین نقطه، **غشای تحتانی**[۶] پیمایش می‌شود.

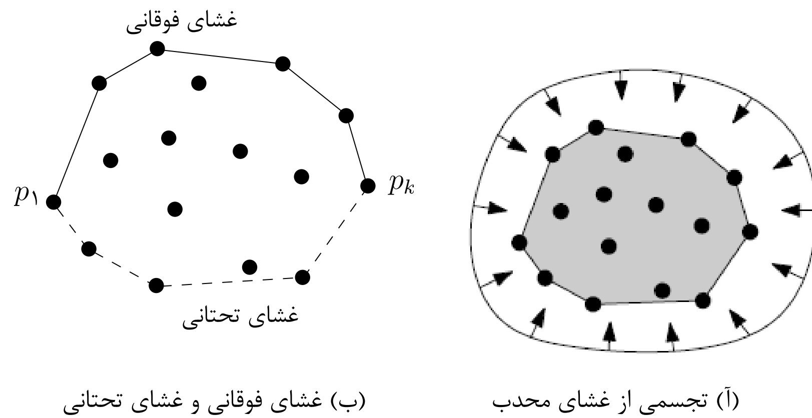
[۳] Convex

[۴] Convex hull

[۵] Upper hull

[۶] Lower hull

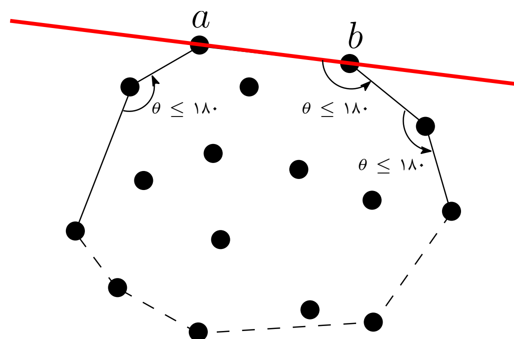
www.jozve.org



شکل ۲.۳: تعریف غشای محدب

مشاهده ۴.۱.۳ پاره‌خط $\overline{ab}$، یک ضلع غشای فوقانی مجموعه نقاط S است اگر و تنها اگر تمام $n-۲$ نقطه باقی‌مانده در یک طرف خط جهت‌دار (ab) (سمت راست، در پیمایش ساعت‌گرد چند ضلعی) قرار بگیرند. شکل ۳.۳ را ببینید.

مشاهده ۵.۱.۳ چنانچه مرز یک چندضلعی دلخواه در جهت ساعت‌گرد پیمایش شود، در هر راس یک چرخش به سمت چپ یا راست وجود دارد. اما در چندضلعی‌های محدب هر چرخش، به سمت راست است و یا به عبارت دیگر زاویه چرخش کمتر از ۱۸۰ درجه است. از آنجایی‌که هر غشای محدب، یک چندضلعی محدب است بنابراین در تمام راس‌های آن گردش به راست وجود دارد. شکل ۳.۳ را ببینید.



شکل ۳.۳: روش تشخیص یک ضلع غشای فوقانی و مفهوم گردش به راست.

قضیه ۶.۱.۳ ([۳۱]) غشای محدب اجتماع دو چندضلعی محدب، در زمانی متناسب با تعداد کل راس‌های این دو چندضلعی قابل محاسبه است.

www.jozve.org

۲.۱.۳ الگوریتم ترتیبی محاسبه غشای محدب

برای محاسبه غشای محدب مجموعه‌ای از نقاط صفحه، الگوریتم‌های ترتیبی مختلفی ارائه شده است که برخی از آن‌ها در مراجع [۵، ۱۳، ۳۱، ۳۸] بیان شده است.

الگوریتم ۴، با استفاده از روش تقسیم و غلبه، غشای محدب مجموعه S شامل N نقطه را محاسبه می‌کند([۳۱]، فصل ۳).

فرض کنید زمان محاسبه غشای اجتماع دو چندضلعی محدب که هر یک شامل $\frac{N}{۲}$ راس هستند، با $U\left(N\right)$

الگوریتم ۴ : محاسبه ترتیبی غشای محدب

ورودی: مجموعه S شامل N نقطه از صفحه

خروجی:$CH\left(S\right) =$ غشای محدب مجموعه نقاط S

۱. اگر $|S| \leq ۳$ باشد، غشای محدب مستقیماً از طریق محاسبه مثلث یا خط گذرنده از نقاط محاسبه شده و الگوریتم خاتمه می‌یابد وگرنه گام ۲ اجرا می‌گردد.

۲. **گام تقسیم:** مجموعه S به دو زیرمجموعه تقریباً مساوی $S_۱$ و $S_۲$ تقسیم می‌شود.

۳. غشای محدب هر یک از مجموعه‌های $S_۱$ و $S_۲$ به‌صورت بازگشتی محاسبه می‌شود.

۴. **گام ترکیب:** دو غشای حاصل از گام قبل، با یکدیگر ترکیب شده و غشای $CH\left(S\right)$ حاصل می‌شود.

نشان داده شود (زمان گام ترکیب). در این‌صورت $T\left(N\right)$ زمان محاسبه کل الگوریتم، برابر با نامساوی ۳-۱ است.

$$T(N) \leq ۲T(\frac{N}{۲}) + U(N). \qquad (۳-۱)$$

چنانچه $P_۱$ چندضلعی حاصل از محاسبه $CH\left(S_۱\right)$ و $P_۲$ چندضلعی حاصل از محاسبه $CH\left(S_۲\right)$ باشند، به ترتیب دارای m و n راس باشند، در این صورت زمان محاسبه گام ترکیب، طبق قضیه ۶.۱.۳ از مرتبه $O\left(m+n\right)$ است. از آنجایی‌که m و n تقریباً برابر $\frac{N}{۲}$ هستند، بنابراین $U\left(N\right)$ برابر با $O\left(N\right)$ است و جواب معادله بازگشتی ۳-۱ برابر رابطه ۳-۲ است.

$$T(N) = N\log N. \qquad (۳-۲)$$

کران پایین یافتن غشای محدب یک مجموعه از N نقطه در صفحه، همانطور که در مرجع [۳۸] اثبات شده

www.jozve.org

است، از مرتبه $\Omega\left(N\log N\right)$ است.

۲.۳ طرح کلی الگوریتم موازی

الگوریتم موازی ارائه شده توسط دیالو[۷]، غشای محدب مجموعه S شامل n نقطه در صفحه را، در مدل CGM محاسبه می‌کند. برای توضیحات بیش‌تر در مورد مدل CGM بخش ۲.۳.۲ را ببینید. ورودی این الگوریتم، مجموعه S شامل n نقطه است. در آغاز، کل مجموعه ورودی در بین حافظه‌های محلی هر یک از پردازنده‌ها توزیع می‌شود و داده‌ها، تا حل شدن کامل مسئله در همان‌جا باقی می‌مانند. الگوریتم تنها به محاسبه غشای فوقانی می‌پردازد و غشای تحتانی و در نتیجه غشای کامل به روشی مشابه قابل محاسبه است. غشای فوقانی مجموعه S با نماد $UH\left(S\right)$ نشان داده می‌شود.

در ادامه فصل، بدون از دست رفتن کلیت مسئله، فرض می‌شود که تمامی نقاط در ربع اول قرار دارند. با این فرض هر جا صحبت از زاویه است، منظور زاویه‌ای است که خطوط گذرنده از نقاط با محور مثبت x-ها می‌سازند. همچنین موقعیت دو پاره‌خط نسبت به یکدیگر بر حسب زاویه‌ای که هر یک با جهت مثبت محور x-ها می‌سازند، سنجیده می‌شود. به عنوان مثال چنانچه پاره‌خط l_1 بالای پاره‌خط l_2 قرار گرفته باشد، به این معنی است که زاویه پاره‌خط l_1 بیشتر از زاویه پاره‌خط l_2 است.

طرح کلی الگوریتم موازی غشای محدب، در الگوریتم ۵ بیان شده است:

تحلیل پیچیدگی:

گام ۱ الگوریتم غشای فوقانی S ، طبق تعریف ۱.۵.۲ با استفاده از یک عمل مرتب‌سازی کلی در زمان $T_s\left(n,p\right)$ اجرا می‌شود.

گام ۲ یک گام کاملاً ترتیبی است و با استفاده از الگوریتم ترتیبی۴ محاسبه می‌شود. $T_{sequuential}$ محاسبه غشای محدب n نقطه، برابر $\Theta\left(n\log n\right)$ است و با توجه به این‌که در حافظه محلی هر یک از پردازنده‌ها، $\frac{n}{p}$ نقطه وجود دارد بنابراین زمان محاسبه ترتیبی غشای محدب بر روی هر یک از پردازنده‌ها برابر رابطه ۳-۳ است.

$$O\left(\frac{n}{p}\log\frac{n}{p}\right)=O\left(\frac{n}{p}\left(\log n-\log p\right)\right)=O\left(\frac{n}{p}\log n-\frac{n}{p}\log p\right)=O\left(\frac{n\log n}{p}\right) \quad (۳-۳)$$

محاسبات هر یک از پردازنده‌ها به صورت موازی و مستقل انجام می‌شود و بنابراین گام ۲، کلاً در زمان

[۷] Diallo

www.jozve.org

الگوریتم ۵ : غشای فوقانی S

ورودی: مجموعه S شامل n نقطه. هر پردازنده $\frac{n}{p}$ نقطه دلخواه از S را در حافظه محلی خود ذخیره می‌کند.
خروجی: نمایش توزیع شده‌ی غشای فوقانی مجموعه S. تمام نقاط روی غشای فوقانی تشخیص داده شده و از چپ به راست برچسب‌گذاری می‌شوند.

۱. نقاط مجموعه S برحسب مولفه x به صورت کلی مرتب می‌شوند. مجموعه $\frac{n}{p}$ نقطه مرتب‌شده که پس از مرتب‌سازی بر روی پردازنده i قرار می‌گیرند، با S_i نشان داده می‌شود.

۲. هر پردازنده i به صورت موازی و مستقل، غشای فوقانی مجموعه S_i را محاسبه می‌کند. غشای فوقانی محاسبه شده توسط پردازنده i، با X_i نشان داده می‌شود.

۳. برای هر غشای فوقانی X_i، $1 \leq i \leq p$، بالاترین خط مماس مشترک فوقانی بین این غشا و همه غشاهای فوقانی بعدی X_j، $i < j \leq p$ محاسبه می‌شود. با استفاده از این خطوط مماس فوقانی، نقاط روی غشای فوقانی S شناسایی و برچسب‌گذاری می‌شوند.

$O\left(\frac{n \log n}{p}\right)$ قابل محاسبه است.

پیچیدگی اصلی این الگوریتم مربوط به **گام ۳** است. در این گام با استفاده از یک الگوریتم ترکیب، p غشای فوقانی با یکدیگر ترکیب شده و غشای نهایی حاصل می‌شود. در ادامه فصل، دو الگوریتم ترکیب موازی شرح داده می‌شوند و ثابت می‌شود که زمان محاسبه هر یک برابر $O\left(\frac{n \log n}{p} + T_s\left(n,p\right)\right)$ است.
در نتیجه با توجه به پیچیدگی زمانی هر سه گام الگوریتم، پیچیدگی زمانی کل الگوریتم غشای فوقانی S برابر با $O\left(\frac{n \log n}{p} + T_s\left(n,p\right)\right)$ است.

۳.۳ ترکیب موازی غشاهای محدب

در این بخش دو الگوریتم ترکیب متفاوت بررسی می‌شود:

۱. الگوریتم $MergeHull$۱ که در قسمت ۶.۳.۳ شرح داده می‌شود، یک الگوریتم ترکیب ساده است که به تعداد ثابتی از دورهای ارتباطی کلی و فضای حافظه $\frac{n}{p} \geq p^2$، برای هر پردازنده احتیاج دارد.

۲. الگوریتم $MergeHull$۲ که در قسمت ۷.۳.۳ شرح داده می‌شود، یک الگوریتم ترکیب پیچیده است

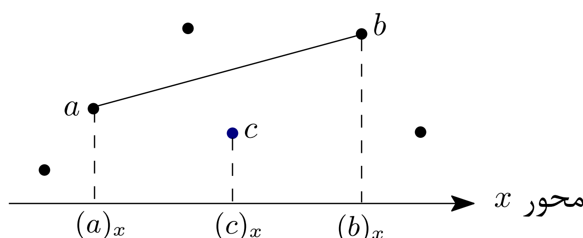
www.jozve.org

که از الگوریتم ترکیب اول به عنوان یک زیر رویه استفاده می‌کند و مقیاس‌پذیری بیشتری دارد به این معنی‌که با حافظه‌های محلی $\frac{n}{p} \geq p^{\epsilon}$، $\epsilon > ۰$ نیز قابل اجراست. با این وجود، این الگوریتم همچنان به تعداد ثابتی دور ارتباطی احتیاج دارد.

۱.۳.۳ تعاریف و مفاهیم مورد نیاز الگوریتم ترکیب

نمادگذاری: پاره‌خطی که دو نقطه a و b را به هم متصل می‌کند با نماد $\overline{ab}$ و خط گذرنده از نقاط a و b با نماد(ab) نشان داده می‌شود. همچنین مختص‌های x و y نقطه a، به ترتیب با نمادهای $(a)_x$ و $(a)_y$ نشان داده می‌شوند.

نقطه c را**مغلوب** پاره‌خط $\overline{ab}$ گویند اگر و تنها اگر $(c)_x$ دقیقاً بین $(a)_x$ و $(b)_x$ قرار گیرد و همچنین c در زیر پاره‌خط $\overline{ab}$ باشد. به عنوان مثال از بین نقاط شکل ۴.۳، تنها نقطه c مغلوب $\overline{ab}$ است.



شکل ۴.۳: نقطه c مغلوب $\overline{ab}$ است.

تعریف ۱.۳.۳ فرض کنید $\{S_i\}$ یک افراز از مجموعه S باشد به قسمی که به ازای هر $j > i$، هر $p \in S_j$ و هر $q \in S_i$، رابطه $(p)_x > (q)_x$ برقرار باشد.

به عنوان مثال در شکل ۵.۳، $S = \{S_۱, S_۲, S_۳\}$ است و نقاط توپر در S_i، عناصر واقع بر روی غشای فوقانی S_i هستند یا به عبارتی دیگر، نقاط توپر غشای X_i را تشکیل می‌دهند.

تعریف ۲.۳.۳ فرض کنید $X_i = \{x_۱, x_۲, \ldots, x_m\}$ یک غشای فوقانی باشد. در این صورت عناصر پیشین[۸] و پسین[۹] $x_j \in X_i$ به صورت رابطه ذیل تعریف می‌شوند:

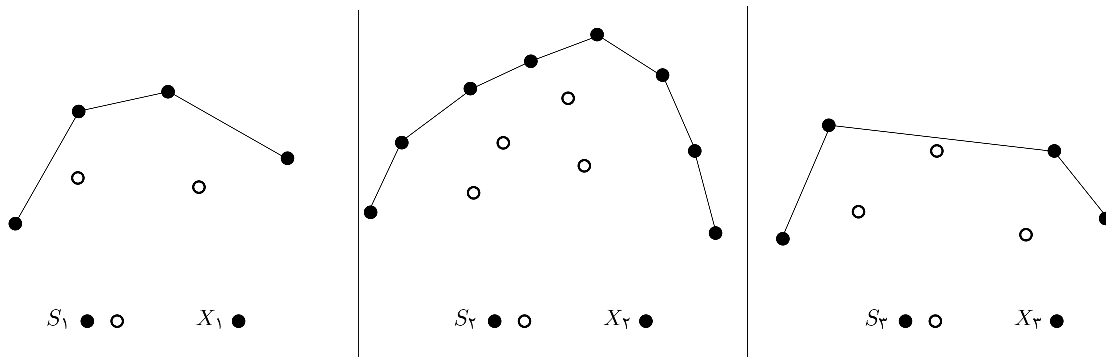
$$pred_{X_i}(x_j) = x_{j-۱}, \quad pred_{X_i}(x_۱) = x_۱$$

$$suc_{X_i}(x_j) = x_{j+۱}, \quad suc_{X_i}(x_m) = x_m$$

[۸] Predecessor

[۹] Successor

www.jozve.org



شکل ۵.۳: افراز مجموعه S به سه زیر مجموعه

مثال ۵.۳.۳ را ببینید.

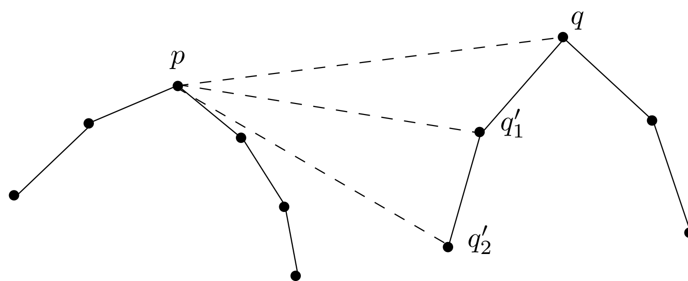
ایده اصلی گام ترکیب الگوریتم ۵

در گام ترکیب، تمام غشاهای X_i که هر یک بر روی پردازنده شماره i ذخیره شده‌اند با یکدیگر ترکیب شده و یک غشای نهایی حاصل می‌گردد. برای هر نقطه q واقع بر روی غشای X_i، تلاش می‌شود تا نقطه‌ای که در غشای نهایی بلافاصله بعد از این نقطه قرار می‌گیرد محاسبه شود. این نقطه به اختصار، عنصر بعدی نامیده می‌شود. عنصر بعدی یا $suc_{X_i}(q)$ است و یا متعلق به یکی از غشاهای X_j ، $j > i$ است.
به عنوان مثال با در نظر داشتن دو غشای فوقانی $X_i = UH(S_i)$ و $X_j = UH(S_j)$ که همه نقاط واقع در S_j، در سمت راست تمامی نقاط S_i قرار دارند؛ عمل ترکیب بر این اساس انجام می‌شود که برای یک نقطه $q \in X_i$، نقطه‌ی $p \in X_i \cup X_j$ که در غشای $UH(X_i \cup X_j)$ بلافاصله بعد از نقطه q قرار خواهد گرفت، محاسبه می‌شود. مثال ۵.۳.۳ را ببینید.
برای تشریح جزئیات گام ترکیب، به تعاریف ذیل نیاز است.

تعریف ۳.۳.۳ فرض کنید $Q \subseteq S$ باشد. در این صورت $Next_Q : S \longrightarrow Q$، تابعی با ضابطه $Next_Q(p) = q$ است اگر و تنها اگر q در سمت راست p قرار داشته باشد و برای تمام $q' \in Q$ که q' در سمت راست p قرار دارد، $\overline{pq}$ بالای $\overline{pq'}$ باشد. شکل ۶.۳ را ببینید. q عنصر بعدی p نامیده می‌شود و در مواردی که مجموعه Q مشخص است اندیس ذکر نمی‌شود.

طبق این تعریف، تابع $Next_Q(p)$، نقطه‌ای مانند $q \in Q$ را مشخص می‌کند که این نقطه در غشای محدب حاصل از ترکیب نقاط $p \cup Q$، بلافاصله بعد از p قرار می‌گیرد. همچنین در سراسر این فصل، منظور از

www.jozve.org



شکل ۶.۳: روش تشخیص عنصر بعدی p

$Next_{\bigcup S_j, j>i}(x)$، عنصر بعدی نقطه x نسبت به تمام مجموعه‌های S_j بعدی است. تمام مجموعه‌های S_j بعدی با نماد $\bigcup S_j, j > i$ نشان داده می‌شود.

اولین نقطه از Y (سمت چپ‌ترین نقطه) که عنصر بعدی آن، دیگر در Y نیست، y^* یا سمت چپ‌ترین نقطه Y نامیده می‌شود. بنابراین $Next(y^*) \neq suc_Y(y^*)$. تعریف ۴.۳.۳ را ببینید.

تعریف ۴.۳.۳ فرض کنید $Y \subseteq S_i$ باشد. در این صورت $lm(Y)$، تابعی با ضابطه $lm(Y) = y^*$ است اگر و تنها اگر y^* سمت چپ‌ترین نقطه در Y باشد به طوری‌که $Next_{Y \cup S_j, j>i}(y^*) \notin S_i$.

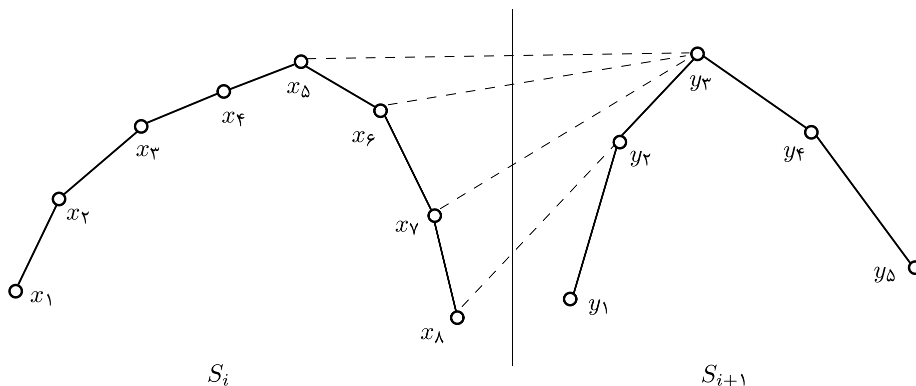
مثال ۵.۳.۳ در شکل ۷.۳، $S' = S_i \cup S_{i+۱}$ مفروض است. در این صورت:

$$x_۳ = Next_{S'}(x_۲) \qquad suc(x_j) = x_{j+۱}$$

$$x_۵ = Next_{S'}(x_۴) \qquad x_۴ = Next_{S'}(x_۳)$$

$$Next_{S'}(x_۸) = y_۲ \text{ ،} \quad Next_{S'}(x_۵) = Next_{S'}(x_۶) = Next_{S'}(x_۷) = y_۳$$

$$lm(S_i) = x_۵$$



شکل ۷.۳: شکل مثال۵.۳.۳.

www.jozve.org

۲.۳.۳ رویه محاسبه عنصر بعدی

فرض کنید X غشای فوقانی یک مجموعه n نقطه‌ای و نقطه c در سمت چپ این مجموعه قرار دارد. رویه $QueryFindNext$ با استفاده از یک جستجوی دودویی، عنصر بعدی نقطه c را نسبت به غشای X محاسبه می‌کند. به این ترتیب که در هر مرحله، مجموعه نقاط X به دو قسمت تقریباً مساوی تقسیم می‌شود و تا پیدا شدن $Next_X\left(c\right)$، جستجو در یکی از قسمت‌ها ادامه می‌یابد. در نتیجه زمان جستجوی این رویه ترتیبی، $O\left(\log\left|X\right|\right)$ است.

رویه ۱: $QueryFindNext\left(X,c,q\right)$

ورودی: غشای فوقانی $X=\{x_۱,\ldots,x_m\}$ که براساس مولفه x مرتب است و نقطه c که در سمت چپ $x_۱$ قرار دارد.

خروجی: نقطه $q\in X$ که q برابر $Next_X\left(c\right)$ است.

۱. اگر $X=\{x\}$ باشد، آنگاه $q\longleftarrow x$ و الگوریتم متوقف می‌شود.

۲. اگر $\overline{x_{\lceil m/۲\rceil}suc\left(x_{\lceil m/۲\rceil}\right)}$ زیر خط $\left(cx_{\lceil m/۲\rceil}\right)$ باشد، آنگاه رویه $QueryFindNext\left(\left\{x_۱,\ldots,x_{\lceil m/۲\rceil}\right\},c,q\right)$ و در غیر این صورت رویه $QueryFindNext\left(\left\{x_{\lceil m/۲\rceil},\ldots,x_m\right\},c,q\right)$ فراخوانی می‌شود.

اثبات درستی رویه ۱:

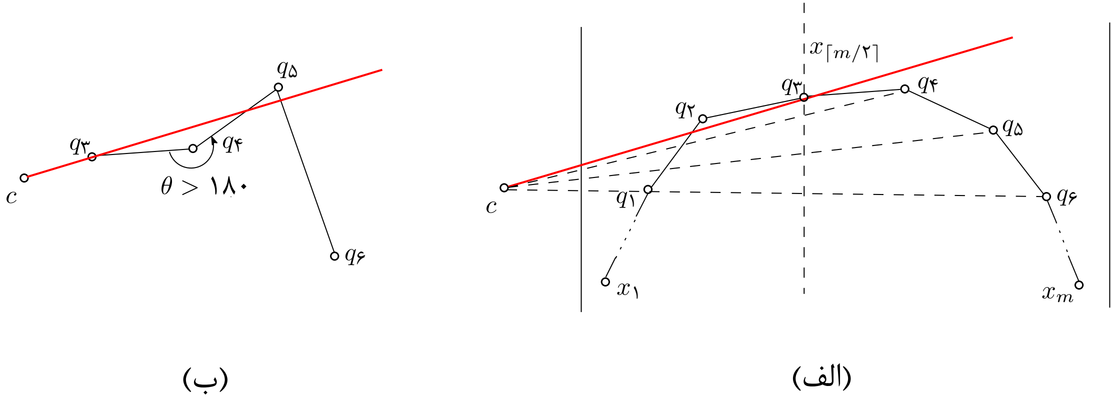
برای اثبات درستی رویه کافی است ثابت کرد که فراخوانی بازگشتی رویه با نصف نقاط ورودی در گام دوم، عنصر بعدی را به درستی محاسبه می‌کند. در شکل‌های ۸.۳ و ۹.۳ جایگذاری‌های ذیل را در نظر بگیرید:

$$q_۳=x_{\lceil m/۲\rceil}\quad,\quad q_۴=suc\left(x_{\lceil m/۲\rceil}\right)$$

در حالت اول که در شکل ۸.۳، نشان داده شده است، $\overline{q_۳q_۴}$ زیر خط $\left(cq_۳\right)$ قرار دارد. در این صورت تمام نقاط بعد از $q_۴$ نیز، در زیر خط $\left(cq_۳\right)$ قرار می‌گیرند. بنا به فرض خلف چنانچه یکی از این نقاط بالای خط $\left(cq_۳\right)$ قرار گیرد، همانطور که در قسمت ب می‌بینید، درست در نقطه ماقبل این نقطه، یک گردش به چپ اتفاق می‌افتد که با خاصیت گردش به راست غشای محدب که در مشاهده ۵.۱.۳ ذکر شد، تناقض دارد. بنابراین تمام پاره‌خط‌های بین نقطه c و هر یک از نقاط زیر خط $\left(cq_۳\right)$ که در شکل به صورت خط‌چین رسم شده است، نیز

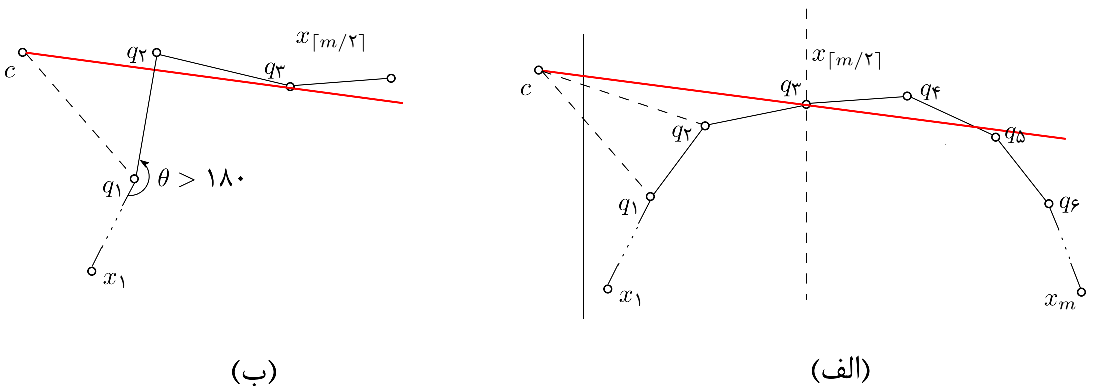
www.jozve.org

در زیر این خط قرار می‌گیرند. طبق تعریف ۳.۳.۳، عنصر بعدی c نمی‌تواند هیچکدام از نقاط زیر خط (cq_3) باشد و فراخوانی بازگشتی باید بر روی نیمه اول نقاط یعنی بازه $\{x_1, \dots, q_3\}$ ادامه یابد.



شکل ۸.۳: اثبات درستی رویه$QueryFindNext$، حالت اول.

شکل ۹.۳، حالت دوم را نشان می‌دهد. در این حالت $\overline{q_3q_4}$ بالای خط (cq_3) قرار می‌گیرد و تمام نقاط قبل از q_3، در زیر این خط (cq_3) قرار می‌گیرند. زیرا همانطور که در قسمت (ب) شکل نشان داده شده است، چنانچه یکی از این نقاط در بالای خط (cq_3) قرار بگیرند، خاصیت گردش به راست غشای محدب از بین خواهد رفت. بنابراین تمام پاره‌خط‌های بین نقطه c و هر یک از نقاط زیر خط (cq_3) که در شکل به صورت خط‌چین رسم شده است، نیز در زیر این خط قرار می‌گیرند. طبق تعریف عنصر بعدی، c نمی‌تواند هیچکدام از نقاط زیر خط (cq_3) باشد و فراخوانی بازگشتی باید بر روی نیمه دوم نقاط یعنی بازه $\{q_3, \dots, x_m\}$ ادامه یابد.



شکل ۹.۳: اثبات درستی رویه$QueryFindNext$، حالت دوم.

www.jozve.org

در نتیجه فراخوانی بازگشتی در هر دو حالت گام دوم رویه، طبق تعریف عنصر بعدی انجام شده و در نهایت عنصر بعدی را به‌درستی محاسبه می‌کند. □

۳.۳.۳ توصیف جدیدی از غشای فوقانی

در الگوریتم موازی غشای فوقانی، ابتدا مجموعه نقاط ورودی S به مجموعه‌های S_i افراز شده و هر پردازنده p_i، $1 \leq i \leq p$، غشای فوقانی X_i را محاسبه می‌کند. در گام ترکیب، هر پردازنده p_i، نقاطی از X_i را که در غشای نهایی باید ظاهر شوند شناسایی می‌کند. شناسایی این نقاط براساس توصیف جدیدی از غشای فوقانی S که S' نامیده می‌شود، انجام می‌شود. این توصیف جدید، طبق خاصیت ضلع غشای فوقانی که در مشاهده ۴.۱.۳ به آن اشاره شد و همچنین تعاریف ۱.۳.۳ تا ۴.۳.۳، تعریف می‌شود.

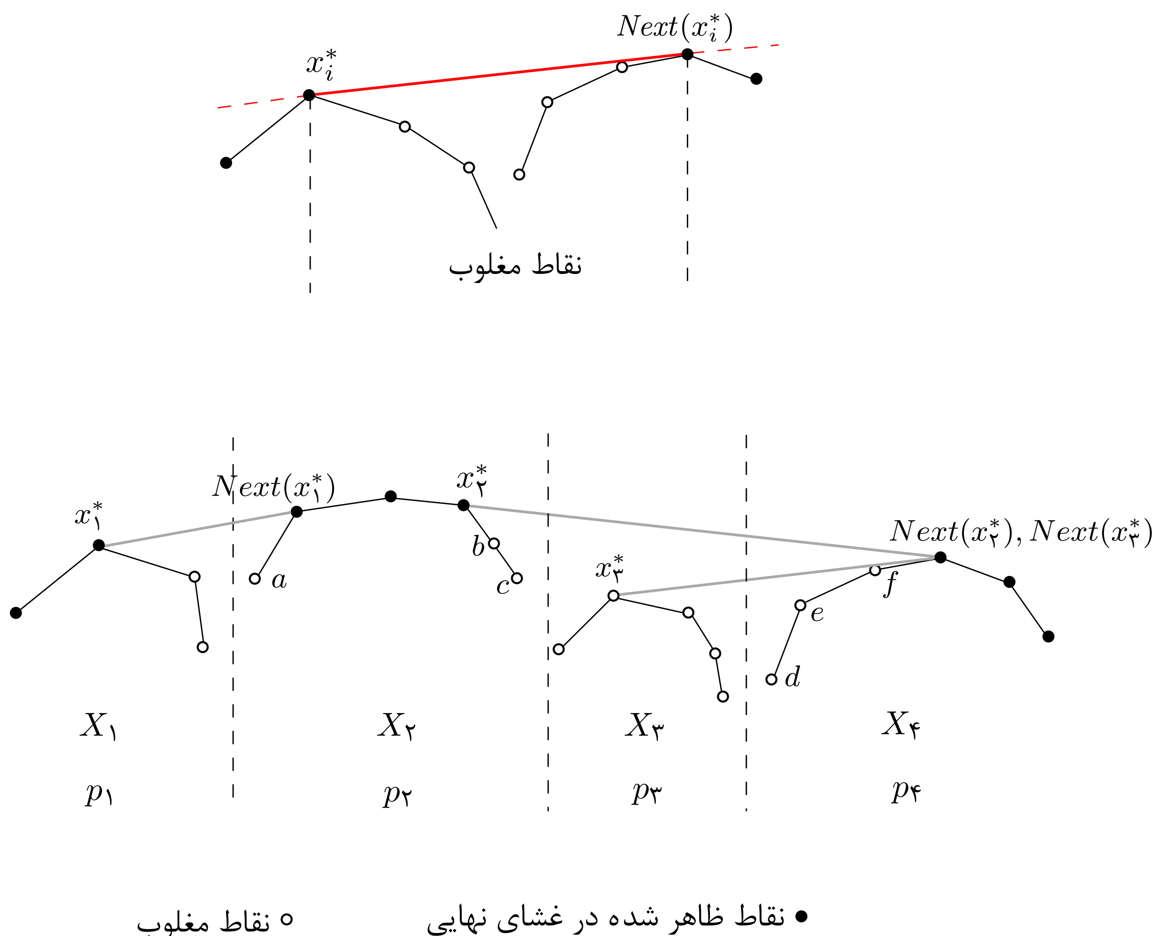
تعریف ۶.۳.۳ مجموعه‌های S، S_i و X_i که در تعاریف ۱.۳.۳ و ۲.۳.۳ معرفی شده‌اند را در نظر بگیرید. مجموعه S' برابر نقاط $c \in \bigcup X_i$ است به طوری که c مغلوب پاره‌خط‌های $\overline{x_i^* Next_{\bigcup X_{j},j>i}\left(x_i^*\right)}$، $1 \leq i < p$، نیست.

در تعریف ۶.۳.۳، x_i^* برابر $lm\left(X_i\right)$ است و عنصر $Next_{\bigcup X_{j},j>i}\left(x_i^*\right)$، عنصر بعدی x_i^* است که با در نظر گرفتن $\bigcup X_j$ یعنی تمام نقاط غشاهای بعدی، محاسبه شده است. فرض کنید $q = Next_S\left(x_i^*\right)$ باشد، در این‌صورت طبق تعریف ۳.۳.۳، q در سمت راست x_i^* است و نمی‌تواند متعلق به غشاهای قبل از X_i باشد. همچنین طبق تعریف ۴.۳.۳، عنصر q متعلق به غشای X_i هم نیست و در واقع q معادل $Next_{\bigcup X_{j},j>i}\left(x_i^*\right)$ است. بنابراین در ادامه فصل عنصرهای $Next_{\bigcup X_{j},j>i}\left(x_i^*\right)$ و $Next_S\left(x_i^*\right)$ معادل هم در نظر گرفته می‌شوند و برای سادگی در نوشتار، با عبارت $Next\left(x_i^*\right)$ نشان داده می‌شوند.
همچنین $\overline{x_i^* Next_{\bigcup X_{j},j>i}\left(x_i^*\right)}$، بالاترین مماس مشترک فوقانی از بین تمام مماس‌های مشترک فوقانی بین X_i و هر یک از غشاهای فوقانی بعد از آن است.

طبق این تعریف، تنها نقاطی از مجموعه $\bigcup X_i$ که توسط **هیچکدام** از پاره‌خط‌های مماس مشترک بالایی مغلوب نشده‌اند، در غشای نهایی ظاهر می‌شوند. زیرا این پاره‌خط‌ها کاندید برای انتخاب ضلع‌های غشای نهایی هستند و طبق مشاهده ۴.۱.۳، برای هر ضلع به غیر از نقاط انتهایی آن، سایر نقاط در زیر خط گذرنده از این ضلع قرار می‌گیرند. نقاطی که مغلوب پاره‌خط شده‌اند، دیگر نمی‌توانند در غشای نهایی ظاهر شوند. شکل ۱۰.۳

www.jozve.org

را ببینید.



شکل ۱۰.۳: عدم حضور نقاط مغلوب $\overline{x_i^* Next\left(x_i^*\right)}$ در غشای نهایی.

بنابراین توصیف S'، برابر غشای فوقانی مجموعه S است.

حقیقت ۷.۳.۳ فرض کنید S، مجموعه ورودی شامل n نقطه در صفحه و S'، مجموعه معرفی‌شده در تعریف ۶.۳.۳ باشد. در این صورت $S' = UH\left(S\right)$ است.

روش شناسایی نقاط مغلوب: پردازنده p_i و غشای $X_i = \{x_1, \ldots, x_i^*, \ldots, x_m\}$ را در نظر بگیرید. فرض کنید که هر پردازنده p_i عنصر $x_i^* = lm\left(X_i\right)$ و $Next_S\left(x_i^*\right)$ مربوط به خود را می‌شناسد و همچنین از مختصات عناصر $Next_S\left(x_j^*\right)$، $i < j$، یعنی عناصر بعدی هر یک از پردازنده‌های قبلی خود مطلع است (پردازنده‌های ما قبل p_i، قبلاً عنصر $Next_S\left(x_i^*\right)$ خود را به p_i ارسال کرده‌اند).

در این صورت پردازنده p_i، می‌تواند پس از بررسی دو حالت زیر، نقاطی از X_i که می‌بایست در غشای نهایی ظاهر شوند را گزارش کند (بررسی‌ها در هر پردازنده همزمان و مستقل از یکدیگر انجام می‌شوند).

www.jozve.org

حالت اول: پردازنده p_i تمام نقاط X_i را که مغلوب $\overline{x_i^* Next\left(x_i^*\right)}$ هستند را نادیده می‌گیرد. به عنوان مثال در شکل ۱۰.۳، پردازنده $p_۲$ نقاط b و c را نادیده می‌گیرد.

حالت دوم: بررسی این حالت، برای هر عنصر بعدی دریافتی از پردازنده‌های ماقبل، انجام می‌شود. فرض کنید عنصر بعدی دریافتی با متغیر q نشان داده شود. پردازنده p_i، مقدار $(q)_x$ را با مقدار $(x_k)_x$ ، $۱ \leq k \leq m$، مقایسه می‌کند:

۱. اگر $(q)_x$ کوچکتر از $(x_۱)_x$ باشد، عملی انجام نمی‌دهد. به عنوان مثال در شکل ۱۰.۳، پردازنده‌های $p_۳$ و $p_۴$ با دریافت عنصر $Next\left(x_۱^*\right)$ هیچ عملی انجام نمی‌دهند.

۲. اگر $(q)_x$ برابر $(x_k)_x$ باشد که در آن $(x_۱)_x \leq (x_k)_x \leq (x_i^*)_x$ است، تمام نقاط قبل از x_k را نادیده می‌گیرد. به عنوان مثال در شکل ۱۰.۳، پردازنده $p_۲$ با دریافت $Next\left(x_۱^*\right)$ نقطه a و پردازنده $p_۴$ با دریافت هر یک از عناصر $Next\left(x_۲^*\right)$ و $Next\left(x_۳^*\right)$ نقاط d، e و f را نادیده می‌گیرند.

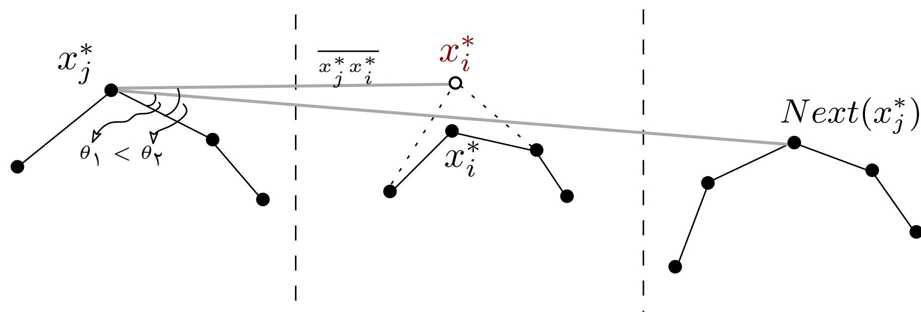
۳. اگر $(q)_x$ بزرگتر از $(x_i^*)_x$ باشد تمام نقاط X_i را نادیده می‌گیرد. به عنوان مثال در شکل ۱۰.۳، پردازنده $p_۳$ تمام نقاط غشای خود یعنی $X_۳$ را نادیده می‌گیرد.

دلیل نادیده گرفتن تمام نقاط X_i: طبق نمادگذاری معرفی شده در قسمت ۱.۳.۳، برای مغلوب شدن نقطه c توسط پاره‌خط $\overline{ab}$، باید دو شرط زیر برقرار باشد:

- مقدار $(c)_x$ دقیقاً بین مقادیر $(a)_x$ و $(b)_x$ قرار گیرد.
- نقطه c در زیر خط (ab) قرار گیرد.

در قسمت ۳، چنانچه q برابر $Next\left(x_j^*\right)$ باشد که j یک اندیس دلخواه کوچک‌تر از i است، از آنجایی‌که $(x_i^*)_x$ کمتر از $(q)_x$ است و از طرفی $(x_i^*)_x < \left(x_j^*\right)_x$ است، بنابراین x_i^* حتماً در محدوده بین x_j^* و $Next\left(x_j^*\right)$ قرار می‌گیرد. از طرفی x_i^* حتماً زیر خط $\left(x_j^* Next\left(x_j^*\right)\right)$ قرار دارد زیرا در غیر این‌صورت، همانطور که در شکل ۱۱.۳ می‌بینید، $\overline{x_j^* x_i^*}$ بالاتر از $\overline{x_j^* Next\left(x_j^*\right)}$ قرار می‌گیرد ($\theta_۲ > \theta_۱$) و طبق تعریف ۳.۳.۳، عنصر بعدی x_j^* برابر x_i^* خواهد بود که یک تناقض است. بنابراین نقطه x_i^* و نقاط قبل از آن مغلوب خواهند بود. نقاط بعد از x_i^* نیز همواره مغلوب پاره‌خط $\overline{x_i^* Next\left(x_i^*\right)}$ هستند. در نتیجه در قسمت ۳، کل نقاط مغلوب شده و پردازنده مربوطه هیچ نقطه‌ای را گزارش نمی‌کند.

www.jozve.org



شکل ۱۱.۳: اثبات درستی قسمت ۳ حالت دوم در شناسایی نقاط مغلوب.

با انجام هر بررسی، مرتباً نقاط بیشتری از X_i نادیده گرفته می‌شوند و نقاط باقی‌مانده نقاطی هستند که می‌بایست گزارش شوند. این نقاط توسط هر پردازنده از چپ به راست برچسب‌گذاری می‌شوند. خروجی نهایی یعنی مجموعه S' برابر است با خروجی هر یک از p پردازنده که از سمت چپ به راست گزارش شده‌اند.

۴.۳.۳ مجموعه جداکننده‌ها و معرفی g_i^*

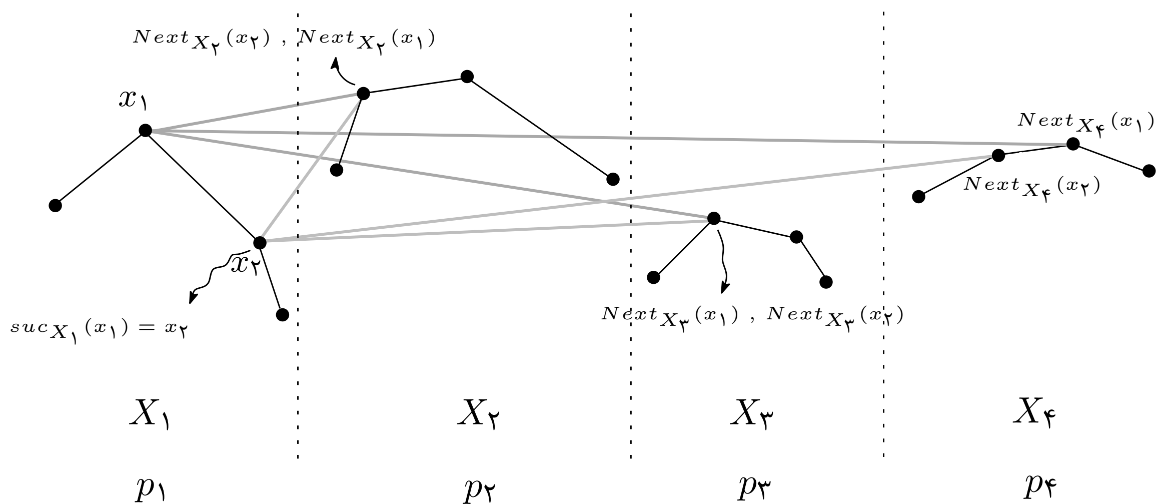
با توجه به توصیف ارائه شده در بخش ۳.۳.۳، اولین قدم محاسبه غشای فوقانی یک مجموعه از نقاط، تعیین نقاط x_i^* و $Next_{\bigcup X_j, j>i}(x_i^*)$ برای هر یک از غشاهای X_i است. نقطه x_i^* طبق تعریف ۴.۳.۳، سمت چپ‌ترین نقطه X_i است که عنصر بعدی آن متعلق به X_i نیست.

برای محاسبه x_i^*، ابتدا عنصر بعدی تمامی نقاط X_i، نسبت به هر یک از غشاهای بعدی محاسبه می‌شود. در نتیجه به ازای هر نقطه روی غشای X_i، به تعداد غشاهای بعد از آن، عنصر بعدی محاسبه شده که از بین آن‌ها، بالاترین عنصر به‌عنوان عنصر بعدی نهایی آن نقطه، انتخاب می‌شود. با یک پویش ساده روی نقاط X_i، می‌توان سمت چپ‌ترین نقطه‌ای که عنصر بعدی آن متعلق به X_i نیست را تعیین نمود. جزئیات نحوه محاسبه در رویه بخش ۵.۳.۳ شرح داده می‌شود.

مثال ۸.۳.۳ در شکل ۱۲.۳ نحوه محاسبه عناصر بعدی غشای $X_۱$ در یک $CGM(۱۶,۴)$ نشان داده شده است. برای محاسبه سه عنصر بعدی $Next_{X_۲}(x_۱)$، $Next_{X_۳}(x_۱)$ و $Next_{X_۴}(x_۱)$ ، لازم است که هر یک از پردازنده‌های $p_۲$، $p_۳$، و $p_۴$ از مختصات نقطه $x_۱$ آگاه باشند. بنابراین پردازنده $p_۱$ مختصات این نقطه را به سه پردازنده بعدی ارسال نموده و هر یک از پردازنده‌های p_j با فراخوانی رویه

www.jozve.org

$QueryFindNext(X_j, x_۱, q)$ عنصر بعدی را محاسبه نموده و نتیجه را به پردازنده $p_۱$ باز می‌گردانند. این پردازنده برای تعیین بالاترین عنصر بعدی نقطه $x_۱$، زاویه پاره‌خط‌های گذرنده از هر یک از عناصر بعدی دریافتی یعنی $\overline{x_۱ Next_{X_j}(x_۱)}$ و پاره‌خط $\overline{x_۱ suc_{X_۱}(x_۱)}$ را محاسبه نموده و بزرگترین زاویه را انتخاب می‌کند. بنابراین عنصر $Next_{X_۲}(x_۱)$ به‌عنوان عنصر بعدی نهایی نقطه $x_۱$ انتخاب می‌شود. روش ذکر شده به طور مشابه و همزمان، برای سایر نقاط $X_۱$ اجرا می‌گردد تا عنصر بعدی نهایی تمامی نقاط $X_۱$ محاسبه شود. با یک پویش ساده روی نقاط $X_۱$، سمت چپ‌ترین نقطه‌ای که عنصر بعدی آن متعلق به $X_۱$ نیست یعنی $x_۱^*$ تعیین می‌شود.



شکل ۱۲.۳: محاسبه عناصر بعدی غشای $X_۱$ در یک $CGM(۱۶, ۴)$.

همانطور که در مثال قبل مشاهده نمودید، برای تعیین عنصر x_i^*، لازم است که تمام نقاط X_i به همه پردازنده‌های بعدی ارسال گردند. چنانچه تعداد نقاط ورودی مسئله محاسبه غشای محدب زیاد باشد، تبادل مجموعه‌های بزرگ X_i بین پردازنده‌ها، سبب می‌شود تا اکثر زمان اجرای الگوریتم موازی، صرف انجام دورهای ارتباطی گردد. برای توضیحات بیشتر در مورد دورهای ارتباطی بخش ۴.۲ را ببینید. بنابراین برای کاهش ارتباطات در مدل موازی، از مجموعه **جداکننده‌ها**[۱۰] استفاده می‌شود. یعنی به جای این‌که رویه $Next$ برای تمام عناصر غشای X_i محاسبه شود، یک زیرمجموعه‌ی مرتب شامل p نقطه با فاصله یکسان، از بین نقاط غشای X_i انتخاب شده و G_i نامیده می‌شود. رویه $Next$ تنها بر روی عناصر این مجموعه اجرا می‌شود و در نهایت

[۱۰] Splitters

www.jozve.org

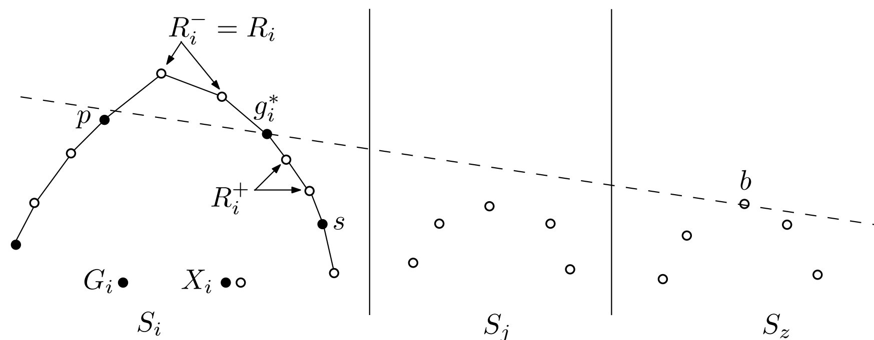
برای هر $g \in G_i$ بزرگترین زاویه از بین پاره‌خط‌های $\overline{gsuc_{G_i}(g)}$ و $\overline{gNext_{X_j,j>i}(g)}$ محاسبه شده تا g_i^* شناسایی شود. با استفاده از نقطه g_i^* می‌توان نقطه x_i^* را محاسبه نمود.

استفاده از جداکننده‌ها این امکان را فراهم می‌کند تا کلیه ارتباطات بین پردازنده‌ها، با مجموعه عملیات ارتباطی ساده که در بخش ۵.۲ معرفی شدند، قابل انجام باشد. در این صورت ارتباط بین پردازنده‌ها، تنها به فراخوانی رویه‌های مناسب از کتابخانه‌های استاندارد خلاصه شده و در نتیجه الگوریتم در عمل، ساده‌تر و سریع‌تر اجرا می‌گردد. برای توضیحات بیشتر در مورد انواع عملیات ارتباطی بخش ۵.۲ را ببینید. هر دو الگوریتم ترکیب ارائه شده در این فصل، از ایده انتخاب مجموعه‌ای از جداکننده‌ها استفاده می‌کنند.

برای تعیین نقطه x_i^* با استفاده از نقطه g_i^*، به تعاریف و لم زیر احتیاج است.

تعریف ۹.۳.۳ فرض کنید $G_i \subseteq X_i$ و نقطه $g_i^* = lm(G_i)$ باشد. در این‌صورت $R_i^- \subseteq X_i$ از نقاط بین $pred_{G_i}(g_i^*)$ و g_i^* تشکیل شده است و $R_i^+ \subseteq X_i$ از نقاط بین g_i^* و $suc_{G_i}(g_i^*)$ تشکیل شده است.

در شکل۱۳.۳، غشای X_i شامل نقاط توپر و توخالی است و مجموعه G_i (جداکننده‌ها) زیرمجموعه‌ی مرتبی از X_i و تنها شامل نقاط توپر می‌باشد. نقطه p برابر $pred_{G_i}(g_i^*)$ و نقطه s برابر $suc_{G_i}(g_i^*)$ است. در این صورت نقاط بین p و g_i^* مجموعه R_i^- و نقاط بین g_i^* و s مجموعه R_i^+ را تشکیل می‌دهند.



شکل ۱۳.۳: مجموعه جداکننده G_i و مجموعه‌های R_i^- و R_i^+.

لم ۱۰.۳.۳ تمام نقاط یک یا هر دو مجموعه‌های R_i^+ و R_i^-، در زیر خط $(g_i^* Next_{X_j,j>i}(g_i^*))$ قرار دارند.

اثبات. (فرض خلف) فرض کنید تمامی نقاط هیچکدام از مجموعه‌های R_i^+ و R_i^-، در زیر خط $(g_i^* b)$ قرار ندارند که در آن b برابر $Next_{X_j,j>i}(g_i^*)$ است. در این‌صورت از هر مجموعه حداقل یک نقطه وجود دارد که در بالای این خط قرار گیرد. نقطه s^+ را اولین نقطه از R_i^+ در نظر بگیرید که طبق فرض خلف در بالای

www.jozve.org

خط (g_i^*b) قرار می‌گیرد و نقطه s^- را آخرین نقطه از R_i^- در نظر بگیرید که طبق فرض، در بالای خط (g_i^*b) قرار می‌گیرد. در این صورت همانطور که در شکل ۱۴.۳ می‌بینید، چهار حالت مختلف ممکن است رخ دهد.

حالت اول: در نقطه $pred_{X_i}(s^+)$ یک گردش به چپ اتفاق می‌افتد.

حالت دوم: در نقطه $suc_{X_i}(s^-)$ یک گردش به چپ اتفاق می‌افتد.

حالت سوم: در نقطه g_i^* یک گردش به چپ اتفاق می‌افتد.

حالت چهارم: در نقاط $pred_{X_i}(s^+)$ و $suc_{X_i}(s^-)$ یک گردش به چپ اتفاق می‌افتد.

در هر یک از چهار حالت بالا، وقوع گردش به چپ با خصوصیت غشای محدب که در مشاهده ۵.۱.۳، ذکر شد تناقض دارد. بنابراین فرض خلف نمی‌تواند صحت داشته باشد و تمام نقاط حداقل یکی از مجموعه‌های R_i^+ و R_i^-، در زیر خط (g_i^*b) قرار دارند. □

تعریف ۱۱.۳.۳ از بین مجموعه‌های R_i^+ و R_i^-، مجموعه‌ای که حداقل یکی از نقاط آن در بالای خط $\left(g_i^*Next_{X_j,j>i}\left(g_i^*\right)\right)$ واقع شده است، R_i نامیده می‌شود. در شکل ۱۳.۳، R_i^+ شامل هیچ نقطه‌ای در بالای خط (g_i^*b) نیست، بنابراین $R_i = R_i^-$ است.

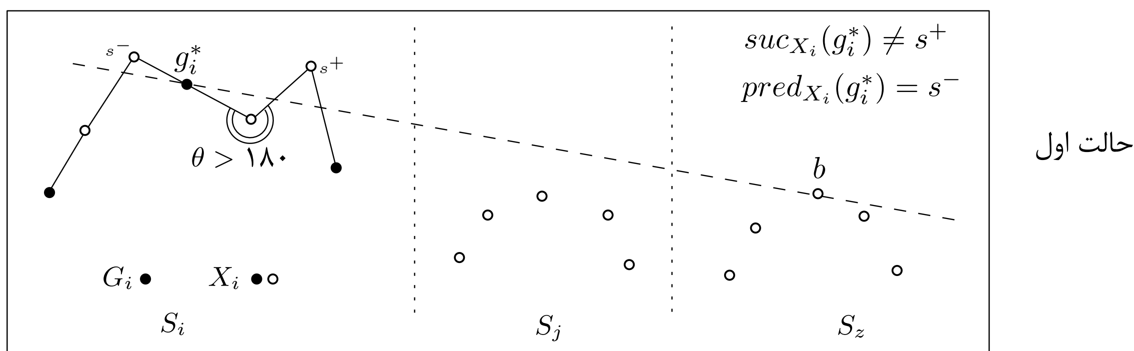
اندازه مجموعه R_i، حداکثر برابر با تعداد نقاطی است که در بین دو نقطه متوالی در G_i قرار می‌گیرند.

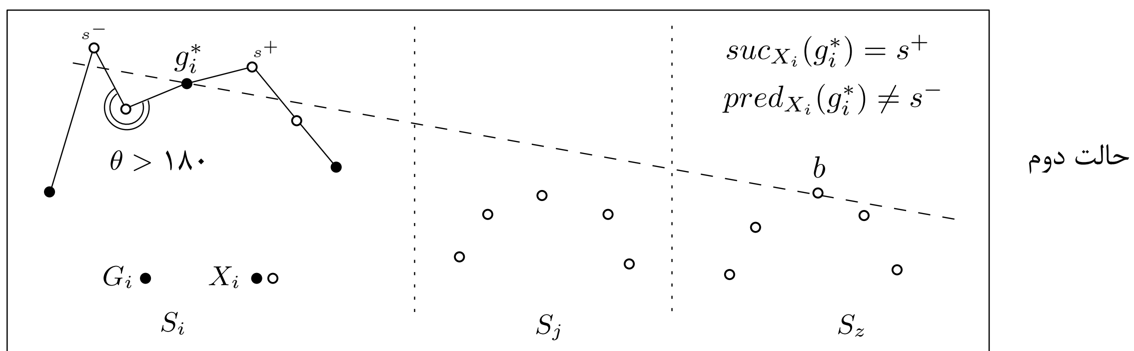
نتیجه ۱۲.۳.۳ با توجه به لم ۱۰.۳.۳ و تعریف ۱۱.۳.۳ می‌توان نتیجه گرفت که $x_i^* = lm\left(R_i \cup g_i^*\right)$ است. زیرا همان‌طور که در شکل ۱۵.۳ قسمت (الف) می‌بینید، وقتی تمام نقاط هر دو مجموعه R_i^+ و R_i^-، در زیر خط (g_i^*b) قرار گیرند عنصر g_i^* همان x_i^* است. در غیر این‌صورت x_i^* از مجموعه‌ای که حداقل یکی از نقاط آن در بالای خط (g_i^*b) است یعنی R_i، انتخاب می‌شود. شکل ۱۵.۳، قسمت (ب) را ببینید. لازم بذکر است که برای یافتن نقطه x_i^* تنها باید دو مجموعه R_i^+ و R_i^- مورد بررسی قرار گیرد و بنا به خصوصیت گردش به راست غشای محدب ممکن نیست که این نقطه بعد از $suc_{G_i}\left(g_i^*\right)$ (در شکل ۱۵.۳، نقطه s) و یا قبل از مجموعه $pred_{G_i}\left(g_i^*\right)$ (در شکل ۱۵.۳، نقطه p) قرار گیرد.

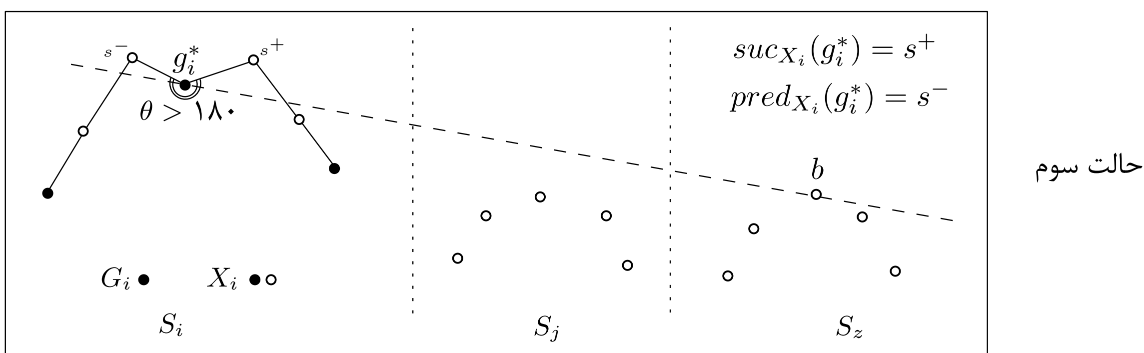
۵.۳.۳ رویه محاسبه g_i^*

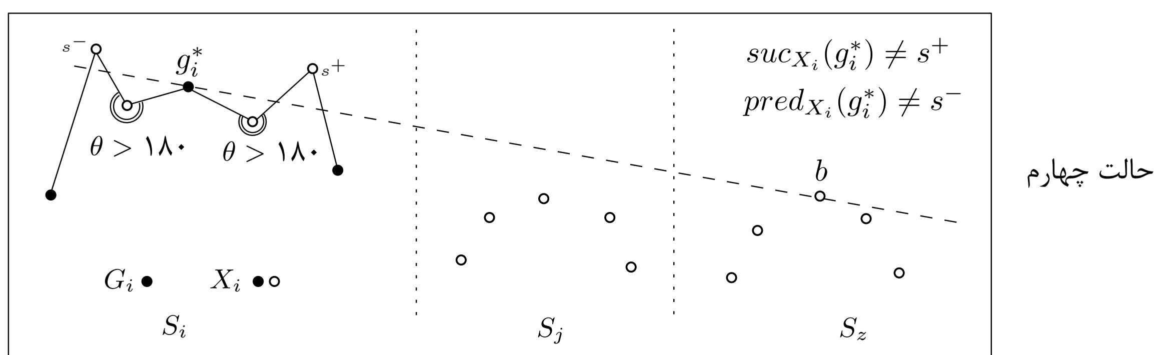
رویه $FindLMSubset$ که در این بخش مورد بررسی قرار می‌گیرد، نحوه محاسبه عنصر g_i^* که برابر $lm\left(G_i\right)$ است را نشان می‌دهد. مجموعه $G_i \subseteq X_i$ مجموعه جداکننده‌های نقاط غشای X_i است که در بخش ۴.۳.۳

www.jozve.org



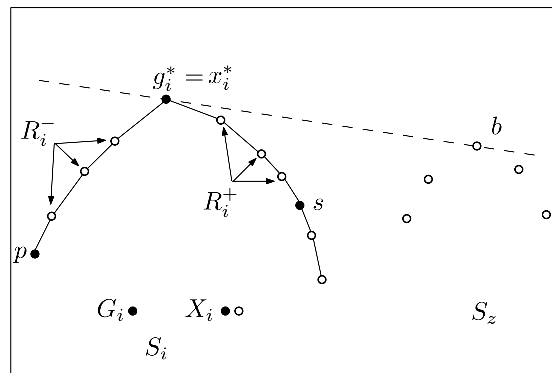




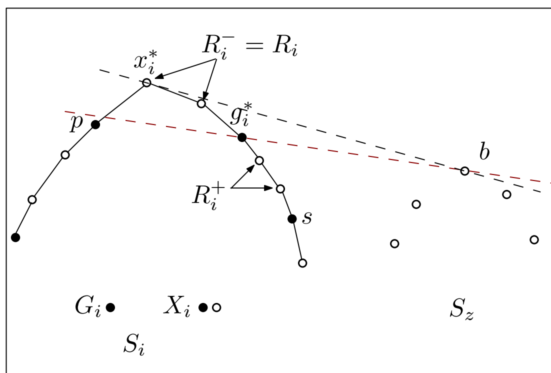


شکل ۱۴.۳: انواع حالات ایجاد شده در صورت برقراری فرض خلف در لم ۱۰.۳.۳.

معرفی شد. روش کلی در این رویه، ارسال عناصر G_i به تمام پردازنده‌های با شماره بیشتر در گام ۲ است به طوری‌که پردازنده‌های p_i که مجموعه $\bigcup_{j<i} G_j$ را دریافت می‌کنند در گام ۳ قادر به محاسبه ترتیبی نقاط مورد



(الف)



(ب)

شکل ۱۵.۳: عنصر x_i^* برابر با $lm\left(R_i \cup g_i^*\right)$ است.

نیاز با استفاده از رویه $QueryFindNext$ هستند. برای توضیحات بیشتر در مورد رویه $QueryFindNext$ بخش ۲.۳.۳ را ببینید. این پردازنده‌ها در گام ۴، پاسخ محاسبه شده را به پردازنده‌های ماقبل خود باز می‌گردانند. سپس هر یک از پردازنده‌ها می‌توانند مستقلاً عنصر g_i^* مربوط به خود را محاسبه کنند.

رویه $FindLMSubset$ توسط هر دو الگوریتم ترکیب مورد بحث این فصل فراخوانی می‌شود و بخش اصلی این الگوریتم‌ها را تشکیل می‌دهد. بنابراین پارامترهای ورودی به‌گونه‌ای انتخاب شده‌اند که هر دو الگوریتم را پوشش دهند. در این رویه پردازنده‌های موجود، به گروه‌هایی تقسیم می‌شوند و پارامتر ورودی k تعیین‌کننده تعداد گروه‌هاست. منظور از q_z^i ، z-امین پردازنده گروه i-ام است که i در رابطه $۱ \leq i \leq p^k$ و z در رابطه $۱ \leq z \leq p^w$ صدق می‌کند. تعداد پردازنده‌ها در هر گروه برابر p^w و در نتیجه تعداد کل پردازنده‌های مورد استفاده در این رویه برابر p^{w+k} است. لیستی از تمام پردازنده‌ها را در نظر بگیرید و فرض کنید که m-امین پردازنده این لیست را با q_m، $۱ \leq m \leq p^{w+k}$ نشان می‌دهند. در این‌صورت چنانچه رابطه زیر برقرار باشد، پردازنده q_z^i همان پردازنده شماره q_m خواهد بود:

$$m = (i - ۱) * p^w + z \qquad (۴\text{-}۳)$$

در تحلیل دورهای ارتباطی رویه، برچسب‌گذاری پردازنده‌ها، براساس موقعیت هر پردازنده در لیست کلی پردازنده‌ها انجام می‌شود و بنابراین به محاسبه شماره q_m نیاز می‌شود.

هر یک از غشاهای محلی با Δ_i نمایش داده می‌شوند که نقاط هر یک از این غشاها، بر روی p^w پردازنده توزیع می‌شوند. به عبارت‌دیگر مجموعه نقاط ذخیره شده بر روی پردازنده‌های گروه i، نمایش توزیع شده‌ای از غشای Δ_i است.

www.jozve.org

رویه ۲ را ببینید. در ادامه، جزئیات اجرای هر یک از گام‌های رویه همراه با یک مثال ساده توضیح داده می‌شود. این مثال، نحوه اجرای رویه را بر روی $CGM\,(۱۶, ۱۲)$ با پارامترهای اولیه $p^k = ۴$ و $p^w = ۳$، نشان می‌دهد و منظور از داده نقاطی است که می‌بایست انتقال یابند.

رویه ۲ : $FindLMSubset\,(\Delta_i, k, w, G_i, g_i^*)$

ورودی: غشاهای فوقانی Δ_i, $۱ \leq i \leq p^k$ که نقاط هر یک بر روی p^w پردازنده با شماره‌های متوالی q_z^i، $۱ \leq z \leq p^w$ توزیع شده‌اند و مجموعه $G_i \subseteq \Delta_i$.

خروجی: نقطه $g_i^* = lm\,(G_i)$. یک کپی از نقطه g_i^* در هر یک از پردازنده‌های q_z^i، $۱ \leq z \leq p^w$ وجود خواهد داشت.

۱. در هر گروه i، نقاط مجموعه G_i به پردازنده $q_۱^i$ ارسال می‌شوند.

۲. هر پردازنده $q_۱^i$، مجموعه G_i خود را به تمام پردازنده‌های q_z^j، $j > i$، $۱ \leq z \leq p^w$ ارسال می‌کند. در نتیجه هر پردازنده q_z^i (برای تمامی i و z ها)، مجموعه $\mathcal{G}^i = \bigcup G_j$ را، برای هر $j < i$ دریافت می‌کند.

۳. هر پردازنده q_z^i (برای تمامی i و z ها)، برای هر $g \in \mathcal{G}^i$ به روش ترتیبی، مقدار $Next_{\Delta_i}\,(g)$ را با استفاده از رویه $QueryFindNext$ محاسبه می‌کند.

۴. هر پردازنده q_z^i (برای تمامی i و z ها)، عنصر $Next_{\Delta_i}\,(g)$ محاسبه شده برای هر $g \in G_j$ را به همه پردازنده‌های $q_۱^j$، $j < i$ باز پس می‌فرستد. در نتیجه هر پردازنده $q_۱^i$ (برای تمامی i ها)، برای هر $g \in G_i$ عنصر $Next_{\Delta_j}\,(g)$ را، برای هر $j > i$ دریافت می‌کند.

۵. هر پردازنده $q_۱^i$ (برای تمامی i ها)، برای هر $g \in G_i$، پاره‌خط با بیشترین زاویه را از بین پاره‌خط‌های $\overline{gsuc_{G_i}\,(g)}$ و $\overline{gNext_{\Delta_j, j>i}\,(g)}$ محاسبه نموده و عنصر $Next_{G_i \cup \Delta_j, j>i}\,(g)$ را پیدا می‌کند. سپس $g_i^* = lm\,(G_i)$ محاسبه می‌شود.

۶. هر پردازنده $q_۱^i$ (برای تمامی i ها)، g_i^* را به q_z^i، $۱ \leq z \leq p^w$ ارسال می‌کند.

گام ۱:

در هر گروه i، نقاط مجموعه G_i که بر روی p^w پردازنده این گروه توزیع شده‌اند، به اولین پردازنده گروه یعنی پردازنده $q_۱^i$ ارسال می‌گردند. شکل ۱۶.۳ را ببینید. این گام، اولین دور ارتباطی رویه است و با یک عمل ارتباطی

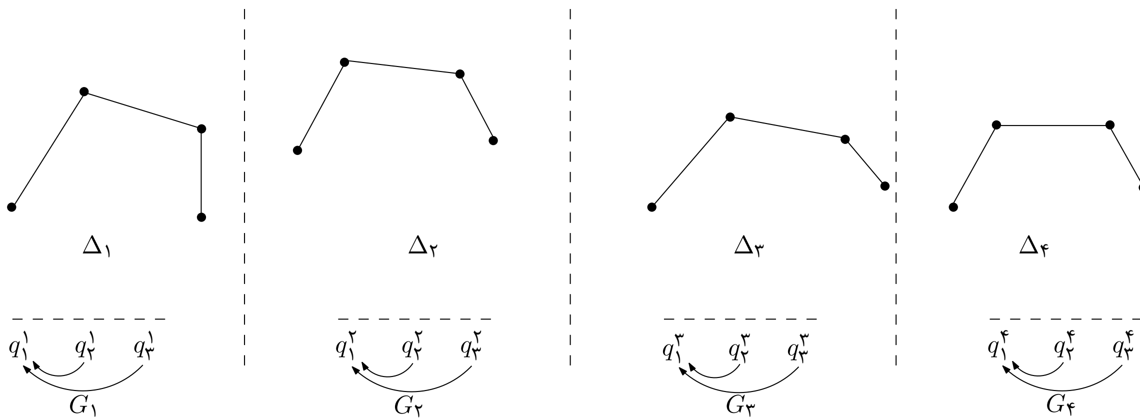
www.jozve.org

تجمیع بسته قابل اجرا است. برای توضیحات بیشتر درباره نحوه پیاده‌سازی عمل تجمیع بسته، اثبات رابطه دوم لم ۷.۵.۲ را ببینید. در پیاده‌سازی این دور ارتباطی برچسب‌گذاری داده‌های هر یک از پردازنده‌های q_z^i، به شکل زیر خواهد بود.

- در پردازنده‌های ارسال کننده یعنی q_z^i، $z \neq ۱$: برچسب قلم‌های داده برابر (m,m) و برچسب قلم‌های خالی برابر (∞,m) است. مولفه m برچسب از رابطه ۳-۴ به دست می‌آید.

- در پردازنده‌های دریافت کننده یعنی $q_۱^i$: برچسب قلم‌های خالی رزرو شده به ترتیب برابر $(m+c_j,m)$، $۱ \leq c_j \leq p^w - ۱$ و برچسب سایر قلم‌های خالی برابر (∞,m) است. لازم به یادآوری است که تعداد قلم‌های خالی رزرو شده برای هر پردازنده دریافت‌کننده با تعداد داده‌هایی که می‌بایست دریافت کند یعنی $p^w - ۱$ برابر است.

پس از این برچسب‌گذاری، عملیات مرتب‌سازی و کپی طبق آنچه در اثبات رابطه دوم لم ۷.۵.۲ بیان شد، انجام می‌شود.



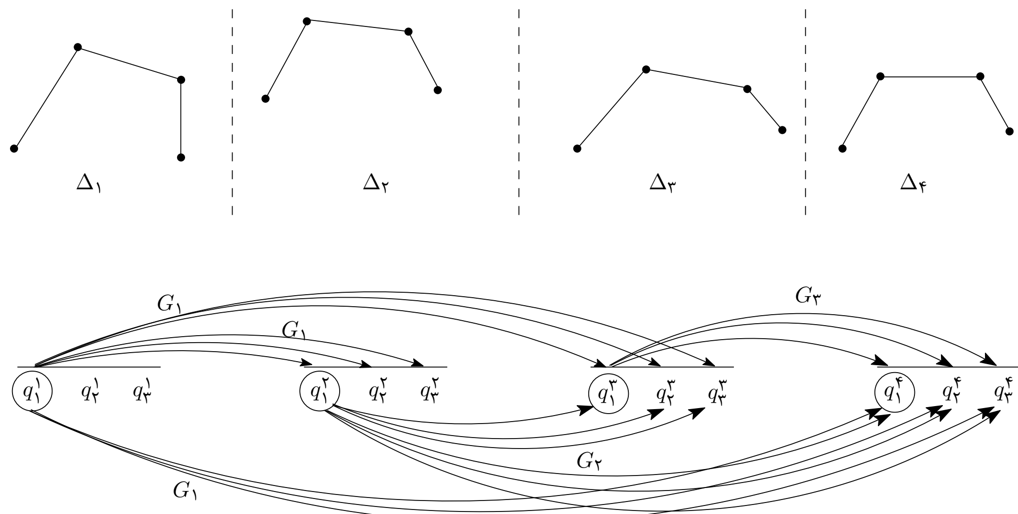
شکل ۱۶.۳: نحوه اجرای گام اول رویه $FindLMSubset$ بر روی $CGM\ (۱۶,۱۲)$.

گام ۲:

هر پردازنده $q_۱^i$ عناصر G_i خود را به تمام پردازنده‌های واقع در گروه‌های بعد از خود ارسال می‌کند. به‌عنوان مثال در شکل ۱۷.۳ پردازنده $q_۱^۱$ مجموعه $G_۱$ را به پردازنده‌های $\{q_۱^۲,q_۲^۲,q_۳^۲\}$ ، $\{q_۱^۳,q_۲^۳,q_۳^۳\}$ و $\{q_۱^۴,q_۲^۴,q_۳^۴\}$ می‌فرستد. پردازنده $q_۱^۲$ مجموعه $G_۲$ را به پردازنده‌های $\{q_۱^۳,q_۲^۳,q_۳^۳\}$ و $\{q_۱^۴,q_۲^۴,q_۳^۴\}$ و پردازنده $q_۱^۳$ مجموعه $G_۳$ را به پردازنده‌های $\{q_۱^۴,q_۲^۴,q_۳^۴\}$ می‌فرستد.

بنابراین هر پردازنده q_z^i، تنها مجموعه داده‌های پردازنده‌های قبل از خود را در قالب یک بسته $\mathcal{G}^i$ دریافت

www.jozve.org



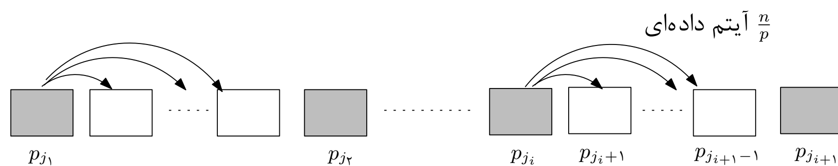
شکل ۱۷.۳: نحوه اجرای گام دوم رویه $FindLMSubset$ بر روی $CGM(16, 12)$.

می‌کند. به‌عنوان مثال در شکل ۱۷.۳ پردازنده‌های گروه چهارم (Δ_4)، داده‌های $\mathcal{G}^4 = G_1 \cup G_2 \cup G_3$ و پردازنده‌های گروه سوم (Δ_3)، داده‌های $\mathcal{G}^3 = G_1 \cup G_2$ را دریافت می‌کند، پردازنده‌های گروه دوم (Δ_2)، تنها داده‌های $\mathcal{G}^2 = G_1$ را دریافت می‌کند و پردازنده‌های گروه اول(Δ_1) در این گام، از هیچ پردازنده‌ای داده دریافت نمی‌کنند. به عبارت دیگر مجموعه $\mathcal{G}^1 = \emptyset$ است.

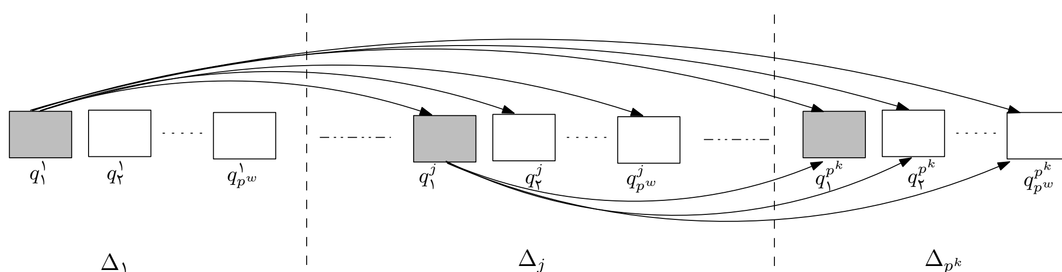
این گام، دومین دور ارتباطی رویه است و با تغییر اندکی در عمل انتشار بسته معرفی شده در تعریف ۲.۵.۲، قابل اجرا است. در عمل انتشار بسته مورد نیاز این گام، لازم است که پردازنده‌های انتخابی، داده‌های خود را به تمام پردازنده‌های گروه‌های بعد از خود، ارسال نمایند در صورتی‌که در تعریف ۲.۵.۲، هر پردازنده داده خود را تنها به تعداد معینی از پردازنده‌ها ارسال می‌کند. برای واضح‌تر شدن این تفاوت، شکل ۱۸.۳ را ببینید. شکل (الف) عمل انتشار بسته را طبق تعریف ۲.۵.۲ نشان می‌دهد و شکل (ب) این عمل را مطابق نیاز گام دوم رویه نشان می‌دهد. برای این‌که عمل انتشار بسته مانند شکل (ب) انجام شود لازم است که برچسب‌های مرتب‌سازی اندکی متفاوت از آنچه در اثبات رابطه اول لم ۷.۵.۲ ذکر شد، تعریف گردند ولی عملیات مرتب‌سازی و کپی همچنان به همان روش انجام خواهند شد. بنابراین در ادامه تنها نحوه برچسب‌گذاری شرح داده می‌شود و از توضیح مجدد کلیدهای مرتب‌سازی و عمل کپی خودداری می‌شود. برای توضیحات بیش‌تر درباره عمل انتشار بسته تعریف ۲.۵.۲ و اثبات رابطه اول لم ۷.۵.۲ را ببینید.

نحوه برچسب‌گذاری: در هر پردازنده به تعداد خانه‌های حافظه آن، قلم داده ایجاد می‌شود. این قلم‌ها، یا داده‌های هستند که می‌بایست منتشر شوند و یا قلم‌های خالی هستند. فرض کنید که تعداد نقاط مجموعه

www.jozve.org



(الف)



(ب)

شکل ۱۸.۳: عمل انتشار بسته مورد نیاز گام دوم رویه $FindLMSubset$.

G_i در تمام پردازنده‌ها یکسان است، در این‌صورت هر پردازنده q_z^i، $i \neq 1$ از پردازنده‌های q_1^j، $j < i$ تعداد $|G_i|$ داده دریافت می‌کند و لازم است که $|G_i| * j$ تا از قلم‌های خالی این پردازنده، جهت دریافت داده‌های پردازنده‌های قبلی، رزرو گردند. این قلم‌ها از سمت چپ به گروه‌های $|G_i|$-تایی تقسیم می‌شوند و مولفه اول برچسب قلم‌های هر گروه برابر q^j، $1 \leq j < i$ خواهد بود. قلم‌های رزرو شده پس از انجام عملیات مرتب‌سازی و کپی، با داده واقعی مورد نظر پر خواهند شد.

قلم‌های هر یک از پردازنده‌ها، به روش زیر برچسب‌گذاری می‌شوند:

(در این برچسب‌گذاری، مولفه m از رابطه ۳-۴ به دست می‌آید و $r' = r \mod |G_i|$ است که عدد r برابر رتبه قلم داده در لیست محلی پردازنده است)

- در پردازنده‌های ارسال کننده داده یعنی q_1^i سه نوع قلم داده وجود دارد (مثال ۱۳.۳.۳ را ببینید):
  - به قلم‌های داده‌ای که می‌بایست منتشر شوند به ترتیب برچسب (q^i, r', m) داده می‌شود.
  - به قلم‌های رزرو شده به ترتیب برچسب (q^j, r', m) داده می‌شود.
  - به سایر قلم‌های خالی باقی مانده، برچسب (∞, r', m) داده می‌شود.
- در پردازنده‌های دریافت کننده داده یعنی q_z^i، $z \neq 1$ دو نوع قلم داده وجود دارد:

www.jozve.org

- به قلم‌های رزرو شده به ترتیب برچسب (q^j, r', m) داده می‌شود.

- به سایر قلم‌های خالی باقی مانده، برچسب (∞, r', m) داده می‌شود.

قلم	توضیح
$(q^{۳}, ۱, ۷)$	آیتم‌های داده‌ای پردازنده $q_۱^۳$ جهت انتشار
$(q^{۳}, ۲, ۷)$	
$(q^{۳}, ۳, ۷)$	
$(q^{۱}, ۱, ۷)$	آیتم‌های رزرو شده برای داده‌های پردازنده $q_۱^۱$
$(q^{۱}, ۲, ۷)$	
$(q^{۱}, ۳, ۷)$	
$(q^{۲}, ۱, ۷)$	آیتم‌های رزرو شده برای داده‌های پردازنده $q_۱^۲$
$(q^{۲}, ۲, ۷)$	
$(q^{۲}, ۳, ۷)$	
$(\infty, r', ۷)$	آیتم‌های بدون استفاده پردازنده $q_۱^۳$

شکل ۱۹.۳: برچسب‌گذاری قلم‌های داده‌ای حافظه پردازنده $q_۱^۳$.

مثال ۱۳.۳.۳ شکل ۱۹.۳، برچسب‌گذاری قلم‌های داده هر یک از خانه‌های حافظه پردازنده $q_۱^۳$ را نشان می‌دهد. فرض کنید که $|G_i| = ۳$ و $p^w = ۳$ است. در این صورت $۱ \leq r' \leq |G_i|$ و $m = ۷$ است.

گام ۳:

نقاط غشای Δ_i بر روی پردازنده‌های q_z^i توزیع شده‌اند. هر پردازنده q_z^i مجموعه $\mathcal{G}^i = \bigcup G_j$، $j < i$ را دریافت می‌کند و به روش ترتیبی، عنصر بعدی هر $g \in G_j$ را نسبت به قسمتی از نقاط Δ_i که در حافظه خود ذخیره دارد، محاسبه می‌کند و سپس بالاترین این عناصر را نشانه‌گذاری می‌کند.

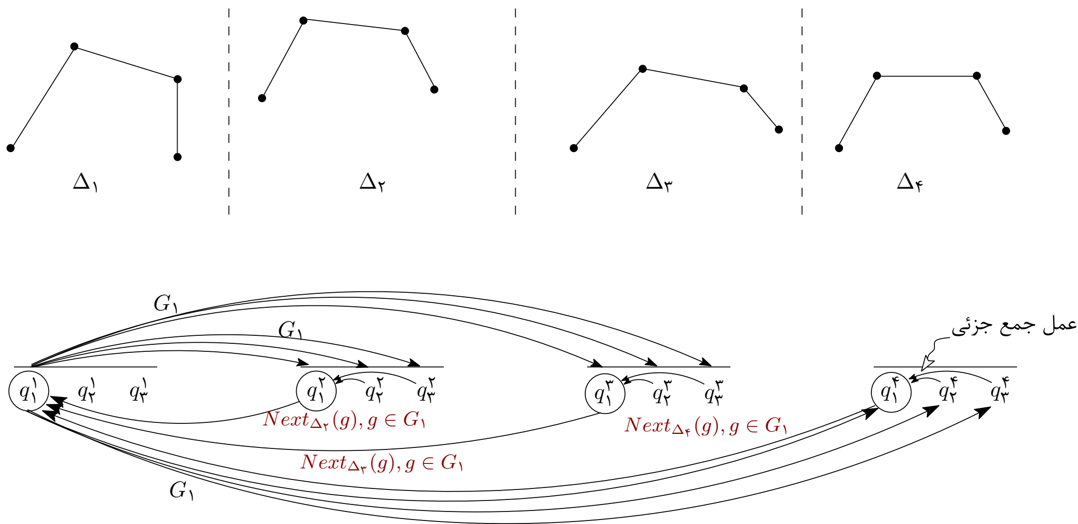
به عنوان مثال هر یک از پردازنده‌های $\{q_۱^۴, q_۲^۴, q_۳^۴\}$ با در اختیار داشتن مجموعه $G_۱$ و با فراخوانی رویه $QueryFindNext\,(\Delta_۴, g, q)$، عناصر بعدی هر $g \in G_۱$ را نسبت به قسمتی از مجموعه $\Delta_۴$ که در حافظه خود ذخیره دارند، به صورت ترتیبی محاسبه می‌کنند و بالاترین عنصر محاسبه شده را نشانه‌گذاری می‌کنند. به همین ترتیب با در اختیار داشتن مجموعه‌های $G_۲$ و $G_۳$، با فراخوانی رویه $QueryFindNext\,(\Delta_۴, g, q)$، عناصر بعدی هر $g \in G_۲$ و هر $g \in G_۳$ را نسبت به قسمتی از مجموعه $\Delta_۴$ که در حافظه خود ذخیره دارند محاسبه می‌کنند و بالاترین عنصر محاسبه شده را نشانه‌گذاری می‌کنند.

www.jozve.org

گام ۴:

در این گام با اجرای یک عمل جمع جزئی در هر گروه i، ابتدا از بین بالاترین عناصر بعدی G_j، $i < j$ که در گام قبل نشانه‌گذاری شده بودند، بالاترین آن‌ها به عنوان عنصر بعدی نهایی نسبت به مجموعه Δ_i محاسبه و در پردازنده سر گروه یعنی q_1^i قرار می‌گیرد و سپس این مقدار به پردازنده‌های q_1^j، $i < j$ ارسال می‌گردد. شکل ۲۰.۳ نحوه اجرای این گام را تنها برای مجموعه G_1 نشان می‌دهد و شکل ۲۱.۳ نحوه اجرای این گام را تنها برای پردازنده‌های گروه چهارم نشان می‌دهد. بنابراین برای هر مجموعه G_j که به گروه Δ_i ارسال می‌گردد، تنها یک عنصر بعدی بازگردانده می‌شود.

این گام، سومین دور ارتباطی رویه است و با یک عمل جمع جزئی و یک عمل تجمیع بسته، قابل اجرا است.

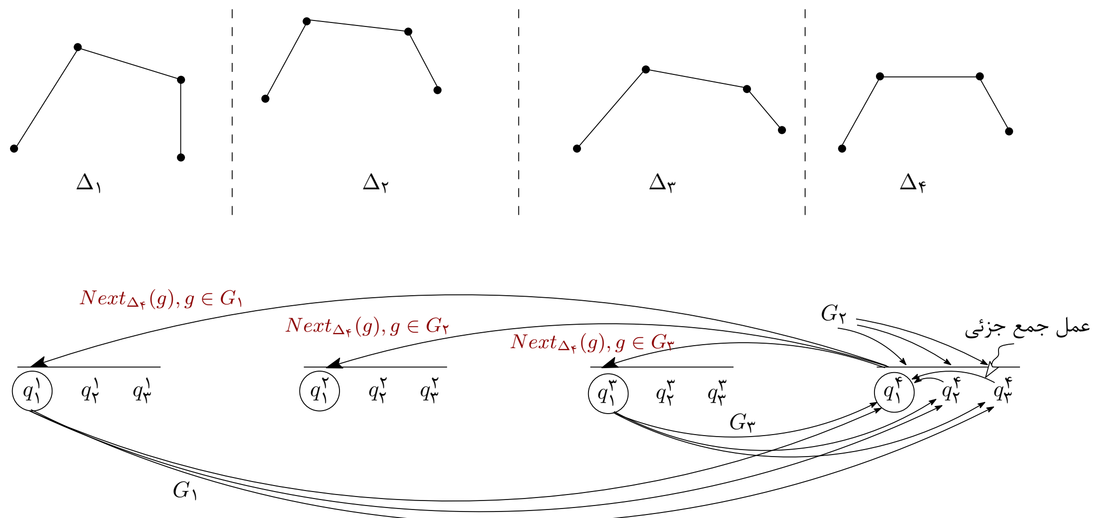


شکل ۲۰.۳: نحوه اجرای گام چهارم رویه $FindLMSubset$ بر روی $CGM\ (16, 12)$، برای G_1.

گام ۵:

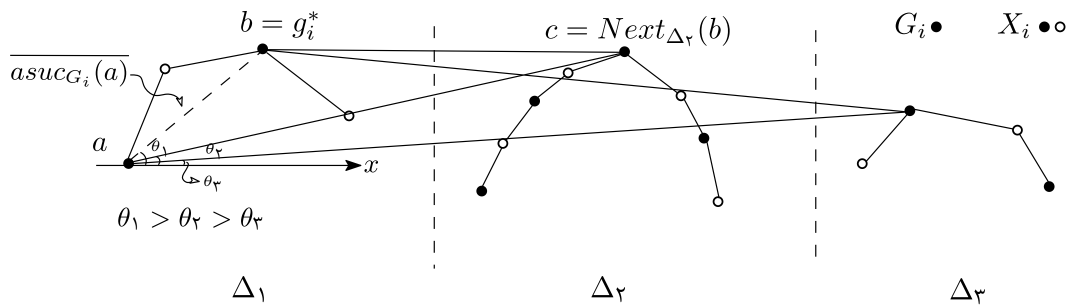
در این گام برای هر عنصر $g \in G_i$، به تعداد گروه‌های بعد از آن، عنصر بعدی محاسبه و موجود است. برای محاسبه نقطه g_i^* از مجموعه G_i، پیمایش از سمت چپ‌ترین نقطه G_i، به عنوان مثال نقطه a در شکل ۲۲.۳، شروع شده و در هر $g \in G_i$، زاویه θ_1 که زاویه پاره‌خط $\overline{gsuc_{G_i}(g)}$ با جهت مثبت محور x است و همچنین زوایایی که پاره‌خط‌های $\overline{gNext_{\Delta_{j,i<j<p}}(g)}$ با جهت مثبت محور x می‌سازند، محاسبه می‌شوند. اگر از بین زوایای محاسبه شده θ_1 بزرگترین زاویه باشد، عنصر بعدی g عنصری از مجموعه X_i است و g نمی‌تواند g_i^* باشد ($g_i^* \notin X_i$)، در غیر این‌صورت (اگر θ_1 بزرگترین زاویه نباشد) g برابر g_i^* است و عنصر بعدی نهایی آن، عنصر بعدی‌ای است که بزرگترین زاویه را با خط افق می‌سازد.

www.jozve.org



شکل ۲۱.۳: نحوه اجرای گام چهارم رویه $FindLMSubset$ در پردازنده‌های گروه $\Delta_۴$.

در مثال شکل ۲۲.۳، نقاط توپر مجموعه G_i را نشان می‌دهند. در نقطه a زاویه $\theta_۱$ با زاویه‌های $\theta_۲$ و $\theta_۳$ مقایسه می‌شود و چون این زاویه بزرگتر است پس a نمی‌تواند عنصر g_i^* باشد، بنابراین محاسبه زوایا و مقایسه آن‌ها در نقطه b تکرار می‌شود. در این نقطه چون زاویه $\theta_۱$ بزرگترین زاویه نیست، نقطه b برابر g_i^* است و عنصر بعدی آن برابر c است.



شکل ۲۲.۳: نحوه محاسبه g_i^* در گام پنجم رویه $FindLMSubset$.

گام ۶:

هر پردازنده $q_۱^i$، g_i^* محاسبه شده را به تمام پردازنده‌های گروه خود (گروه Δ_i) می‌فرستد. این گام، چهارمین دور ارتباطی رویه است و با یک عمل انتشار کلی، قابل اجرا است.

لم ۱۴.۳.۳ رویه $FindLMSubset$ عنصر $g_i^* = lm\left(G_i\right)$ را در تعداد ثابتی از دورهای ارتباطی محاسبه می‌کند. علاوه بر این فضای حافظه محلی مورد نیاز برای هر یک از پردازنده‌ها حداکثر $p^k\left|G_i\right|$ می‌باشد.

www.jozve.org

اثبات. درستی الگوریتم $FindLMSubset$ ناشی از تعاریف ۳.۳.۳ و ۴.۳.۳ و رویه ۱ می‌باشد. از آنجایی‌که این الگوریتم دقیقاً براساس این تعاریف بنا شده است، در نهایت عنصر g_i^* را به درستی محاسبه می‌نماید.

پیچیدگی فضایی گام‌های الگوریتم: در گام ۱ به هر پردازنده، $|G_i|$ داده تخصیص داده می‌شود. پس حداکثر فضای مورد نیاز در این گام، $|G_i|$ می‌باشد. در گام ۲ هر پردازنده، $|G_i|$ داده از پردازنده‌های q_1^i ماقبل خود دریافت می‌کند. بنابراین پردازنده‌های آخرین گروه، حداقل از $p^k - 1$ گروه قبل از خود، داده دریافت می‌کنند و حداکثر فضای مورد نیاز آن‌ها برابر است با:

$$\left|\mathcal{G}^i\right| = |G_1| + \cdots + \left|G_{p^k-1}\right| = O\left(p^k\,|G_i|\right)$$

در گام ۴ هر پردازنده q_1^i، به تعداد گروه‌های بعد از خود، یک عنصربعدی دریافت می‌کند. بنابراین پردازنده q_1^1، حداکثر $p^k - 1$ عنصر را دریافت می‌کند.

در نتیجه فضای مورد نیاز کل الگوریتم از مرتبه $O\left(p^k\,|G_i|\right)$ خواهد بود.

پیچیدگی زمانی گام‌های ترتیبی الگوریتم: در گام ۳ الگوریتم، عنصر بعدی هر یک از عناصر $g \in \mathcal{G}^i$، با فراخوانی رویه $QueryFindNext\left(\Delta_i, g, q\right)$ محاسبه می‌شود که پیچیدگی زمانی این الگوریتم ترتیبی همان‌طور که در بخش ۳.۳ شرح داده شد، $O\left(\log|X_i|\right)$ می‌باشد. X_i مجموعه نقاط روی غشا Δ_i را نمایش می‌دهد. از آنجایی‌که تعداد نقاط مجموعه S به طور مساوی بین هر یک از پردازنده‌ها تقسیم می‌شوند، بنابراین:

$$|X_i| \leq |S_i| = n/p \Longrightarrow \log\left(|X_i|\right) \leq \log\left(|S_i|\right) = \log\left(n/p\right).$$

بنابراین عنصر بعدی هر یک از عناصر مجموعه $\mathcal{G}^i$ در زمان $O\left(\log\left(n/p\right)\right)$ محاسبه می‌شود و تعداد کل عناصر بعدی که می‌بایست محاسبه گردند، $|\mathcal{G}^i| = p^k\,|G_i|$ است. در نتیجه کل زمان اجرای این گام برابر رابطه زیر است:

$$p^k\,|G_i|\,O\left(\log\left(n/p\right)\right) = p^k\,|G_i|\,O\left(\log n - \log p\right) = O\left(p^k\,|G_i|\log n\right)$$

در گام ۵ الگوریتم، حداکثر برای تمامی عناصر G_i، می‌بایست p^k زاویه محاسبه شده و در نهایت بزرگترین زاویه انتخاب شود. هر زاویه در زمان $O(1)$ قابل محاسبه است و بنابراین محاسبه $p^k\,|G_i|$ زاویه، در زمان $O\left(p^k\,|G_i|\right)$ صورت می‌گیرد.

بنابراین پیچیدگی زمانی گام‌های ترتیبی الگوریتم از مرتبه $O\left(p^k\,|G_i|\log n\right)$ خواهد بود.

پیچیدگی زمانی ارتباطات: همانطور که در توضیح گام‌های الگوریتم ذکر شد، گام‌های اول، دوم، چهارم و ششم، چهار گام ارتباطی این رویه هستند. □

www.jozve.org

۶.۳.۳ الگوریتم $MergeHull$۱

در این بخش با استفاده از مفهوم جداکننده‌ها، توصیف جدید غشای فوقانی و همچنین با استفاده از رویه $FindLMSubset$ الگوریتمی ارائه می‌گردد که p غشای فوقانی X_i را با یکدیگر ترکیب نموده و نقاط غشای نهایی را به‌عنوان خروجی گزارش می‌دهد. در ادامه به معرفی این الگوریتم پرداخته و در انتهای بخش، الگوریتم براساس سه فاکتور ارزیابی که در بخش ۴.۲ معرفی شدند، مورد بررسی و ارزیابی قرار می‌گیرد.
الگوریتم ۶ را ببینید. این الگوریتم تنها به ۵ دور ارتباطی احتیاج دارد که این دورهای ارتباطی در سه مرحله انجام می‌شوند.

مرحله ۱: هر پردازنده حداکثر p داده خود را، به عنوان مجموعه جداکننده به پردازنده‌های با شماره بیشتر ارسال می‌کند (در حالتی که تعداد نقاط غشای محدب محلی کمتر از p باشد، کمتر از p داده ارسال می‌گردد). عناصر بعدی مورد نیاز محاسبه شده و بازگردانده می‌شوند.

مرحله ۲: هر پردازنده داده‌های بین دو جداکننده منتخب را به پردازنده‌های با شماره بیشتر ارسال می‌کند. عناصر بعدی مورد نیاز محاسبه شده و بازگردانده می‌شوند.

مرحله ۳: هر پردازنده عنصر $Nexts$ خود را به پردازنده‌های با شماره بیشتر ارسال نموده و سپس هر پردازنده بر اساس اطلاعات دریافتی، نقاطی از غشای خود را که می‌بایست در غشای نهایی ظاهر گردند، شناسایی می‌کند.

لم ۱۵.۳.۳ الگوریتم $MergeHulls$۱ ، $UH\left(S\right)$ را در زمان $O\left(\frac{n\log n}{p}+T_s\left(n,p\right)\right)$ محاسبه می‌کند. این الگوریتم به فضای حافظه محلی $\frac{n}{p}\geq p^{۲}$ و تعداد ثابتی دور ارتباطی احتیاج دارد (عبارت $\frac{n}{p}\geq p^{۲}$ یعنی این‌که اندازه فضای حافظه محلی هر پردازنده که با $\frac{n}{p}$ نشان داده می‌شود حداقل برابر $p^{۲}$ است).

اثبات. **درستی الگوریتم:** گام‌های ۲ و ۴ الگوریتم، طبق لم ۱۴.۳.۳ و نتیجه ۱۲.۳.۳، عناصر g_i^* و x_i^* را به درستی محاسبه می‌کند. گام ۳ الگوریتم دقیقاً براساس تعاریف ۹.۳.۳ ، ۱۱.۳.۳ و لم ۱۰.۳.۳ بنا شده است؛ بنابراین، مجموعه R_i را به درستی محاسبه می‌کند. بر اساس حقیقت ۷.۳.۳، مجموعه‌های S'_i محاسبه

www.jozve.org

الگوریتم ۶: $MergeHulls۱ \left(X_i \left(۱ \leq i \leq p\right), S, n, p, UH\right)$

ورودی: مجموعه‌ای از p غشای فوقانی X_i که در مجموع شامل n نقطه از S هستند و هر غشای X_i بر روی پردازنده q_i ($۱ \leq i \leq p$) ذخیره شده است.

خروجی: نمایش توزیع شده‌ای از غشای فوقانی S . تمام نقاط روی غشای فوقانی تشخیص داده شده و از چپ به راست برچسب‌گذاری می‌شوند.

۱. هر پردازنده q_i، مجموعه جداکننده G_i که شامل p نقطه با فاصله‌های یکسان از مجموعه X_i است را، تعیین می‌کند.

۲. پردازنده‌ها به صورت موازی و همزمان عنصر $g_i^* = lm\left(G_i\right)$ را محاسبه می‌کنند. این محاسبه از طریق فراخوانی رویه $FindLMSubset$ انجام می‌شود.

۳. هر پردازنده q_i با استفاده از عنصر g_i^* محاسبه شده، مجموعه‌های R_i^- و R_i^+ خود را طبق تعریف ۹.۳.۳ محاسبه می‌کند. همچنین مجموعه R_i که شامل نقاطی است که در بالای خط $\left(g_i^* Next_{G_i \cup X_j, j>i}\left(g_i^*\right)\right)$ قرار می‌گیرند، طبق تعریف ۱۱.۳.۳ و لم ۱۰.۳.۳ تعیین می‌شود.

۴. پردازنده‌ها به صورت موازی و همزمان عنصر $x_i^* = lm\left(R_i \cup g_i^*\right)$ را محاسبه می‌کنند. این محاسبه از طریق فراخوانی رویه $FindLMSubset$ انجام می‌شود. عنصر x_i^* سمت چپ‌ترین نقطه در $R_i \cup g_i^*$ است به طوری‌که $Next_S\left(x_i^*\right) \notin X_i$.

۵. هر پردازنده q_i عنصر $Next_S\left(x_i^*\right)$ خود را، به همه پردازنده‌های بعدی ارسال می‌کند و مجموعه S_i' را طبق تعریف ۶.۳.۳ محاسبه می‌کند.

شده در گام ۵، هر یک به درستی نمایش توزیع شده‌ای از $UH\left(S\right)$ می‌باشند و هر یک از پردازنده‌ها با استفاده از روشی که در انتهای بخش ۳.۳.۳ ذکر شد، قادر به شناسایی نقاط مجموعه S_i' هستند.

تحلیل پیچیدگی فضای الگوریتم: این الگوریتم رویه $FindLMSubset$ را دو بار فراخوانی می‌کند و طبق لم ۱۴.۳.۳ فضای مورد نیاز این رویه برابر $p^k\left|G_i\right|$ است.

در گام دوم، این رویه با پارامترهای ورودی زیر فراخوانی می‌شود:

$$\Delta_i = X_i, k = ۱, w = \circ, G_i = G_i$$

www.jozve.org

که اندازه $|G_i|$ برابر p است. پس در این رویه، هر پردازنده p نقطه را به پردازنده‌های بعدی ارسال می‌کند که باعث می‌شود پردازنده آخر $p\,(p-۱)$ داده، از پردازنده‌های ماقبل دریافت کند. در نتیجه حداقل فضای مورد نیاز هر پردازنده، $p^۲$ است. در گام چهارم همین رویه با پارامترهای:

$$\Delta_i = X_i, k = ۱, w = ۰, G_i = R_i \cup g_i^*$$

فراخوانی می‌شود که اندازه $|G_i|$ در رابطه زیر صدق می‌کند:

$$|G_i| = |R_i \cup g_i^*| = \frac{|X_i|}{p} \xrightarrow{|X_i| \le \lceil \frac{n}{p} \rceil} |G_i| \le \frac{n}{p^۲} \qquad (۵\text{-}۳)$$

بنابراین حداکثر اندازه مجموعه $|G_i|$ در این گام برابر $\frac{n}{p^۲}$ است و حداکثر فضای مورد نیاز این گام، برابر $\frac{n}{p}$ خواهد بود. در گام ۵، هر پردازنده عنصر بعدی پردازنده‌های قبل از خود را دریافت می‌کند، پس پردازنده آخر حداقل به فضای p احتیاج خواهد داشت. با توجه به فضای مورد نیاز برای هر گام، کل الگوریتم با حداقل فضای $p^۲$ و حداکثر فضای $\frac{n}{p}$ قابل اجراست.

تحلیل پیچیدگی زمانی الگوریتم: در گام اول الگوریتم، با یک پویش ساده روی نقاط هر یک از لیست‌های محلی، مجموعه جداکننده‌ها تعیین می‌شوند. این پویش به زمان $O\left(\frac{n}{p}\right)$ احتیاج دارد. طبق لم ۱۴.۳.۳ پیچیدگی زمانی اجرای رویه $FindLMSubset$ برابر $O\left(p^k\,|G_i|\log n\right)$ است. بنابراین زمان اجرای گام‌های دوم و چهارم الگوریتم به ترتیب برابر:

$$p^۲\log n \quad \text{و} \quad p\,|R_i \cup g_i^*|\log n = \frac{n}{p}\log n \ \text{(طبق رابطه ۳-۵)}$$

است. در گام سوم با یک پویش ساده روی هر یک از لیست‌های محلی، نقاط بین g_i^* و عناصر پسین و پیشین آن شناسایی شده و مجموعه R_i تعیین می‌شود. این پویش از مرتبه $\frac{n}{p}$ است. در گام پنجم، هر پردازنده عنصر بعدی هر یک از پردازنده‌های قبلی را دریافت می‌کند. در نتیجه پردازنده آخر حداکثر p عنصر را دریافت می‌کند. هر پردازنده با استفاده از روشی که در انتهای بخش ۳.۳.۳ ذکر شد، هر یک از عناصر دریافتی را با دو عنصر $x_۱$ و x_i^* مقایسه می‌کند تا نقاطی را که نباید در غشای نهایی ظاهر گردند، تعیین کند. بنابراین حداکثر مقایسه‌های مورد نیاز این گام $۲p$ است. سپس هر پردازنده با یک پویش ساده روی کل نقاط لیست محلی خود، نقاطی را که در غشای نهایی ظاهر می‌گردند، برچسب‌گذاری می‌کند. بنابراین زمان اجرای کل گام‌های الگوریتم از مرتبه $O\left(\frac{n}{p}\log n\right)$ خواهد بود.

www.jozve.org

تحلیل تعداد فازهای ارتباطی مورد نیاز: طبق لم ۱۴.۳.۳، رویه $FindLMSubset$ برای اجرا تنها به چهار دور ارتباطی احتیاج دارد. در این رویه فرض شده است که هر غشای X_i بر روی p^w پردازنده توزیع شده است و بنابراین در گام اول آن، می‌بایست عناصر مجموعه G_i از روی p^w پردازنده جمع‌آوری و به پردازنده اول ارسال گردند. همچنین در گام آخر رویه، عنصر g_i^* محاسبه شده، می‌بایست به تمام پردازنده‌های گروه i ارسال گردد. برای توضیحات بیشتر در مورد این دو گام بخش ۵.۳.۳ را ببینید. فراخوانی این رویه با پارامترهای $w = 0$ و $k = 1$ در الگوریتم $MergeHulls1$، سبب می‌شود تا رویه تنها به دو فاز ارتباطی احتیاج داشته باشد. زیرا با فرض $w = 0$، تمام نقاط غشای X_i، بر روی تنها پردازنده هر گروه ذخیره شده و دیگر نیازی به دورهای ارتباطی گام‌های اول و آخر رویه نیست. بنابراین فراخوانی رویه $FindLMSubset$ در گام‌های دوم و چهارم الگوریتم $MergeHulls1$، سبب می‌شود تا در مجموع ۴ دور ارتباطی هزینه گردد.

گام پنجم الگوریتم نیز، شامل یک گام ارتباطی است که عبارتند از: ارسال عنصر $Nexts_S\left(x_i^*\right)$ هر پردازنده به تمام پردازنده‌های بعدی. این دور ارتباطی قابل کاهش به عمل ارتباطی انتشار کلی است با این تفاوت که به جای این‌که هر پردازنده داده خود را به تمام پردازنده‌های دیگر ارسال کند، تنها باید به پردازنده‌های بعدی خود ارسال نماید. بنابراین نحوه برچسب‌گذاری ذکر شده در اثبات رابطه (۳) لم ۷.۵.۲ به‌صورت زیر تغییر می‌یابد ولی نحوه مرتب‌سازی به همان شکل باقی خواهد ماند.

نحوه برچسب‌گذاری: هر پردازنده p_i به تعداد $p - i$ کپی از داده خود ایجاد کرده و به هر یک به‌ترتیب برچسب i تا p می‌دهد. همچنین $i - 1$ قلم خالی با برچسب ۱ تا $i - 1$ ایجاد می‌کند و به سایر خانه‌های خالی پردازنده برچسب i تعلق می‌گیرد.

بنابراین الگوریتم در کل با ۵ دور ارتباطی قابل انجام است و طبق لم ۱۴.۳.۳، زمان اجرای هر دور ارتباطی برابر $O\left(\frac{n}{p} + T_s\left(n,p\right)\right)$ است.

□

طبق تحلیل زمانی الگوریتم $MergeHull1$، تمام محاسبات محلی از مرتبه $O\left(\frac{n}{p}\log n\right)$ است و بنابراین الگوریتم از لحاظ فاکتور میزان محاسبات محلی بهینه است. تعداد دورهای ارتباطی مورد نیاز الگوریتم نیز ثابت است و طبق لم ۷.۵.۲، هر دور در زمان $O\left(\frac{n}{p} + T_s\left(n,p\right)\right)$ اجرا می‌شود که چنانچه از یک الگوریتم مرتب‌سازی موازی بهینه استفاده شود، پیاده‌سازی فازهای ارتباطی الگوریتم نیز، بهینه خواهد شد. این الگوریتم از لحاظ فاکتور مقیاس‌پذیری بهینه نیست زیرا تنها برای نسبت‌های $\frac{n}{p} \geq p^2$ قابل اجراست. برای توضیحات

www.jozve.org

بیشتر در مورد فاکتورهای ارزیابی الگوریتم‌های دانه‌درشت، بخش ۴.۲ را ببینید.

الگوریتم $MergeHull۲$ معرفی شده در بخش ۷.۳.۳، به‌نحوی طراحی شده است که علاوه بر دارا بودن بهینگی در فاکتورهای میزان محاسبات محلی و میزان دورهای ارتباطی، مقیاس‌پذیر نیز است.

۷.۳.۳ الگوریتم $MergeHull۲$

الگوریتم $MergeHull۲$ تعداد p غشای فوقانی که هر یک بر روی یکی از p پردازنده CGM ذخیره شده‌اند را به عنوان ورودی دریافت کرده و با استفاده از تعداد ثابتی از دورهای ارتباطی آن‌ها را با یکدیگر ترکیب نموده تا غشای نهایی حاصل گردد. برخلاف الگوریتم $MergeHull۱$، این الگوریتم تنها به $\frac{n}{p} \geq p^{\epsilon}$ فضای حافظه در هر پردازنده احتیاج دارد که $۰ < \epsilon \leq ۱$ است. به عنوان مثال و برای وضوح بیشتر در توصیف الگوریتم، فرض کنید که $\epsilon = ۱$ است. در این‌صورت الگوریتم به صورت زیر عمل می‌کند:

در **فاز اول**، کل پردازنده‌ها به $\sqrt{p}$ گروه که هر گروه شامل $\sqrt{p}$ پردازنده است، تقسیم می‌شوند و از الگوریتم $MergeHulls۱$ استفاده می‌شود تا غشاهای فوقانی هر گروه ترکیب شوند. این کار تنها در پنج دور ارتباطی انجام می‌شود. اثبات لم ۱۵.۳.۳ را ببینید. در الگوریتم $MergeHulls۱$ چنانچه اندازه نمونه $|G_i|$ برابر $\sqrt{p}$ انتخاب شود، پردازنده آخر هر گروه حداکثر p داده را دریافت می‌نماید و در نتیجه فضای مورد احتیاج این فاز برابر $O\left(p\right)$ است. در انتهای این فاز، $\sqrt{p}$ غشای فوقانی حاصل می‌گردد که نقاط هر یک، بر روی $\sqrt{p}$ پردازنده پخش‌اند. چنانچه فرض شود که $\frac{n}{p}$ نقطه ذخیره شده بر روی هر یک از پردازنده‌ها، عیناً در غشای نهایی حاصل از ترکیب غشاهای هر گروه ظاهر گردند، در این‌صورت بر روی غشای حاصل از هر گروه، حداکثر $\frac{n}{p}\sqrt{p}$ نقطه وجود خواهد داشت.

در **فاز دوم**، $\sqrt{p}$ غشای فوقانی حاصل از فاز اول که اندازه هر یک، حداکثر $\frac{n}{p}\sqrt{p}$ است، با یکدیگر ترکیب می‌شوند تا غشای نهایی حاصل گردد. شکل ۲۳.۳ را ببینید. برای ترکیب، مانند روش الگوریتم $MergeHulls۱$، از هر غشا مجدداً $\sqrt{p}$ نقطه، به‌عنوان نمونه G_i انتخاب می‌شود و سپس رویه $FindLMSubset$ فراخوانی می‌شود. به علت توزیع نقاط G_i بر روی $\sqrt{p}$ پردازنده، با یک دور ارتباطی این نقاط در پردازنده اول هر گروه جمع‌آوری شده و سپس به تمام پردازنده‌های گروه بعدی ارسال می‌گردند. پردازنده‌های ذخیره کننده غشای $\sqrt{p}$-ام، حداکثر p نقطه را دریافت می‌کنند و بنابراین g_i^*، با حداقل فضای $O\left(p\right)$ قابل محاسبه است. در اجرای گام آخر رویه نیز، می‌بایست عنصر g_i^* محاسبه شده، به تمام پردازنده‌های گروه i ارسال گردد. بنابراین اجرای این رویه، به چهار دور ارتباطی نیاز دارد. از آنجایی‌که اندازه مجموعه R_i در

www.jozve.org

فاز اول $\left\{ \underbrace{\Phi_1 \cdots \Phi_{\sqrt{p}}} \quad \cdots \quad \underbrace{\Phi_{(j-1)\sqrt{p}+1} \cdots \Phi_{j\sqrt{p}}} \quad \cdots \quad \underbrace{\Phi_{(\sqrt{p}-1)\sqrt{p}+1} \cdots \Phi_p} \right.$

فاز دوم $\left\{ \underbrace{\Psi_1 \quad \cdots \quad \Psi_j \quad \cdots \quad \Psi_{\sqrt{p}}}_{\Psi_{Final}} \right.$

شکل ۲۳.۳: محاسبه ترکیب p غشای فوقانی با فضای حافظه $p \geq \frac{n}{p}$ در هر پردازنده.

بدترین حالت برابر $\frac{n}{p} = \frac{n\sqrt{p}}{p|G_i|}$ است، بنابراین اجرای رویه $FindLMSubset$ بر روی این مجموعه به منظور محاسبه x_i^*، به فضای حداکثر $\frac{n\sqrt{p}}{p}$ نیاز دارد، که خیلی زیاد است. بنابراین به کاهش اندازه بیشتری نیاز است. از آنجایی‌که هر یک از مجموعه‌های R_i، خود یک غشای فوقانی هستند، می‌توان به صورت بازگشتی روشی مشابه آنچه در $MergeHulls1$ استفاده شده است را بر روی مجموعه R_i به کار برد تا اندازه R_i کاهش یابد و بتوان $x_i^* = lm\left(R_i \cup g_i^*\right)$ را با حداقل فضای $O\left(p\right)$ محاسبه نمود. برای این کار نمونه $\sqrt{p}$ تایی G_i' از مجموعه R_i انتخاب می‌شود و با فراخوانی رویه $FindLMSubset$ عنصر $g_i'^*$ محاسبه می‌شود، از آنجایی‌که نقاط R_i تنها بر روی یک پردازنده قرار دارند، نیازی به دورهای ارتباطی گام‌های اول و آخر این رویه نیست و رویه تنها با دو دور ارتباطی انجام می‌شود. پس از فراخوانی این رویه مجموعه R_i جدید، مجدداً محاسبه شده که اندازه آن برابر $\frac{n}{p\sqrt{p}}$ است. بنابراین این‌بار فراخوانی رویه $FindLMSubset$ بر روی مجموعه R_i جدید، به حداکثر فضای $\frac{n}{p}$ نیاز دارد که قابل انجام است. پس از محاسبه x_i^*، در یک دور ارتباطی، عنصر بعدی x_i^* برای محاسبه S_i به پردازنده‌های بعدی ارسال می‌گردد. بنابراین فاز دوم در کل، با هفت دور ارتباطی قابل پیاده‌سازی است.

از ایده‌ای که در حالت $\epsilon = 1$ به‌کار رفته است استفاده می‌شود تا الگوریتم $MergeHull2$ که حالت کلی $\frac{n}{p} \geq p^{\epsilon}$ را پوشش می‌دهد، توصیف گردد. این الگوریتم در هر فاز رویه $BuildHulls$ را فراخوانی می‌کند. رویه ۳ را ببینید. این رویه مشابه الگوریتم $MergeHull1$ عمل می‌کند و در آن پردازنده‌ها به گروه‌هایی تقسیم می‌شوند و q_z^i نشان‌دهنده z–امین پردازنده گروه i–ام است. در ادامه به شرح و تحلیل این رویه پرداخته می‌شود.

www.jozve.org

رویه ۳ : $BuildHulls\left(\Phi_i, \Psi_j, p, k, w, \epsilon\right)$

ورودی: غشاهای فوقانی Φ_i، $۱ \leq i \leq \frac{p}{p^w}$؛ نقاط هر غشا بر روی p^w پردازنده متوالی q_z^i، $۱ \leq z \leq p^w$، توزیع شده است. پارامترϵ اندازه حافظه محلی هر یک از p پردازنده موجود را نشان می‌دهد.

خروجی: غشاهای فوقانی $\Psi_j = UH\left(\bigcup_{i=(j-۱)p^k+۱}^{jp^k} \Phi_i\right)$ برای مقادیر $۱ \leq j \leq \frac{p}{p^{w+k}}$.

۱. $G_i \longleftarrow \Phi_i$.

۲. تا زمانی که رابطه $|G_i|\, p^k > p^\epsilon$ برقرار است، انجام بده:

(آ) هر گروه از پردازنده‌های q_z^i، $۱ \leq z \leq p^w$، یک مجموعه جداکننده G_i' که از $p^{\frac{\epsilon}{۲}}$ نقطه با فاصله‌های یکسان از مجموعه G_i تشکیل شده است، را تعیین می‌کنند (نقاط مجموعه G_i' به طور متناسب بر روی p^w پردازنده گروه i، توزیع شده است).

(ب) $FindLMSubset\left(\Phi_i, k, w, G_i', g_i^*\right)$.

(ج) اگر $\overline{g_i^* suc_{\Phi_i}\left(g_i^*\right)}$ بالای $\overline{g_i^* Next_{\Phi_j, j>i}\left(g_i^*\right)}$ قرار دارد، R_i از همه نقاط Φ_i که بین g_i^* و $suc_{G_i'}\left(g_i^*\right)$ قرار می‌گیرند، تشکیل می‌شود؛ وگرنه R_i از همه نقاط Φ_i که بین $pred_{G_i'}\left(g_i^*\right)$ و g_i^* قرار می‌گیرند، تشکیل می‌شود.

(د) $G_i \longleftarrow R_i \cup \{g_i^*\}$.

۳. $FindLMSubset\left(\Phi_i, k, w, G_i, x_i^*\right)$.

۴. هر پردازنده $q_۱^i$ عنصر $Next_S\left(x_i^*\right)$ خود را، به همه پردازنده‌های q_j^h ارسال می‌کند که $h = i+۱, \ldots, i+\left(p^k - i \mod p^k\right)$ و $۱ \leq j \leq p^w$. چنانچه $\left(i \mod p^k\right)$ برابر صفر باشد، پردازنده i داده‌ای را ارسال نمی‌کند.

۵. هر پردازنده مجموعه S_i' مربوط به خود را طبق تعریف ۶.۳.۳ محاسبه می‌کند.

لم ۱۶.۳.۳ رویه $BuildHulls$ غشاهای $\Psi_j = UH\left(\bigcup_{i=(j-۱)p^k+۱}^{jp^k} \Phi_i\right)$، $۱ \leq j \leq \frac{p}{p^{w+k}}$ را در زمان $O\left(\frac{n \log n}{p} + T_s\left(n,p\right)\right)$ محاسبه می‌کند. با فرض $\alpha = max\left\{\left(k+\frac{\epsilon}{۲}\right), \epsilon\right\}$، این رویه به فضای حافظه محلی $\frac{n}{p} \geq p^\alpha$ و تعداد ثابتی دور ارتباطی احتیاج دارد.

www.jozve.org

اثبات. **پیچیدگی فضایی گام‌های رویه:** در گام ۲(ب)، رویه $FindLMSubset$ با اندازه نمونه $|G_i| = p^{\frac{\epsilon}{2}}$ فراخوانی می‌شود. در اجرای این رویه، غشاهای موجود در دسته‌های p^k-تایی گروه‌بندی شده که $1 \leq k < 0$ است و محاسبات رویه بر روی هر گروه مستقلاً و به طور همزمان انجام می‌شود. در نتیجه فضای مورد نیاز این گام طبق لم ۱۴.۳.۳ برابر $p^{k+\frac{\epsilon}{2}} \leq \frac{n}{p}$ است. در هر تکرار حلقه گام ۲، اندازه مجموعه R_i مرتباً کاهش می‌یابد تا بالاخره فضای مورد نیاز جهت اجرای رویه $FindLMSubset$ در گام سوم برابر p^{ϵ} گردد یا به عبارت دیگر $|G_i|\, p^k \leq p^{\epsilon}$ گردد. در نتیجه فضای مورد نیاز گام سوم برابر $p^{\epsilon} \leq \frac{n}{p}$ است. در گام چهارم نیز پردازنده آخر هر گروه، حداکثر p^k عنصر بعدی را، از پردازنده‌های واقع در گروه خود دریافت می‌کند. بنابراین فضای مورد احتیاج این گام برابر $p^k \leq \frac{n}{p}$ است و فضای مورد نیاز کل رویه برابر p^{α} است که مقدار α برابر $max\left\{\left(k + \frac{\epsilon}{2}\right), \epsilon\right\}$ است.

پیچیدگی زمانی ارتباطات: همانطور که در اثبات پیچیدگی فضایی نشان داده شد حداقل فضایی که لازم است هر پردازنده داشته باشد تا رویه قابل اجرا باشد برابر p^{α} است و از آنجایی‌که هر Φ_i بر روی p^w پردازنده ذخیره شده است، بنابراین در Φ_i حداکثر $p^{\alpha+w}$ نقطه وجود دارد. در گام یک، G_i برابر Φ_i می‌شود و تعداد نقاط G_i در هر گذر با فاکتور $p^{\frac{\epsilon}{2}}$ کاهش می‌یابد. بنابراین به تعداد ثابت:

$$\log_{p^{\frac{\epsilon}{2}}} p^{\alpha+w} = \frac{2\left(\alpha + w\right)}{\epsilon} \qquad (۳-۶)$$

فاز وجود دارد که هر فاز، خود شامل یک‌بار فراخوانی رویه $FindLMSubset$ است و این رویه طبق لم ۱۴.۳.۳ تنها به تعداد ثابتی دور ارتباطی احتیاج دارد. بنابراین در مجموع گام دوم در تعداد ثابتی از دور ارتباطی انجام می‌شود. همچنین گام‌های سوم و چهارم نیز به ترتیب به تعداد چهار و یک دور ارتباطی احتیاج دارند. در نتیجه کل رویه $BuildHulls$ در تعداد ثابتی از دور ارتباطی قابل اجراست. زمان پیاده‌سازی هر دور ارتباطی طبق لم ۷.۵.۲، برابر $O\left(\frac{n}{p} + T_s\left(n, p\right)\right)$ است.

پیچیدگی زمانی گام‌های ترتیبی الگوریتم: پیچیدگی زمانی گام ۳ طبق اثبات لم۱۴.۳.۳، برابر $O\left(p^{\epsilon} \log n\right)$ و پیچیدگی زمانی گام ۲(ب) برابر

$$O\left(|G_i|\, p^k \log n\right) = O\left(\frac{n}{p} \log n\right)$$

است. از آنجایی‌که گام ۲(ب) به تعداد ثابتی تکرار می‌گردد، بنابراین زمان پیچیدگی کل رویه برابر $O\left(\frac{n}{p} \log n\right)$ است.

www.jozve.org

□

الگوریتم ۷ : $MergeHulls2\left(X_i, S, n, p, UH\left(S\right), \epsilon\right)$

ورودی: مجموعه‌ای از p غشای فوقانی X_i که در مجموع حداکثر n نقطه از مجموعه S را شامل می‌شوند و هر غشای X_i بر روی یک پردازنده p_i، $1 \leq i \leq p$، ذخیره شده است. پارامتر ϵ نشان‌دهنده اندازه حافظه‌های محلی است.

خروجی: نمایش توزیع شده‌ای از غشای فوقانی S. همه نقاط روی غشای فوقانی شناسایی شده و از چپ به راست برچسب‌گذاری می‌شوند.

۱. جایگذاری‌های $k = \frac{\epsilon}{2}$، $w = 0$ و $X_i^0 = X_i$ را انجام دهید.

۲. تا زمانی که $w \geq 1$ نیست، انجام بده:

(آ) $BuildHulls\left(X_i^w, X_j^{w+k}, p, k, w, \epsilon\right)$

(ب) $w \longleftarrow w + k$

قضیه ۱۷.۳.۳ الگوریتم $MergeHulls2$، مجموعه $UH\left(S\right)$ را در زمان $O\left(\frac{n \log n}{p} + T_s\left(n, p\right)\right)$ محاسبه می‌کند. این الگوریتم به فضای حافظه محلی $\frac{n}{p} \geq p^\epsilon$ و تعداد ثابتی دور ارتباطی احتیاج دارد.

اثبات. **پیچیدگی فضایی الگوریتم:** طبق لم ۱۶.۳.۳ فضای مورد نیاز جهت اجرای رویه $BuildHulls$ در گام ۲ (آ)، برابر $\frac{n}{p} \geq p^\alpha$ است که با توجه به این‌که $k = \frac{\epsilon}{2}$ است، α برابر ϵ است و فضای مورد نیاز این الگوریتم برابر $\frac{n}{p} \geq p^\epsilon$ است.

پیچیدگی زمانی ارتباطات: پارامتر w با مقدار اولیه صفر در هر تکرار حلقه، به اندازه $k = \frac{\epsilon}{2}$ واحد افزایش می‌یابد تا این‌که مقدار آن برابر یک گردد. در این‌صورت اجرای حلقه متوقف می‌شود. بنابراین تعداد تکرارهای حلقه یا به عبارتی دیگر تعداد فراخوانی‌های رویه $BuildHulls$ برابر $\frac{2}{\epsilon}$ است و طبق رابطه ۳-۶، در هر فراخوانی

www.jozve.org

تعداد $\frac{۲(\epsilon+w)}{\epsilon}$ فاز وجود دارد. در گذر t-ام حلقه الگوریتم $MergeHulls$۲، مقدار پارامتر w در گام ۲(آ) برابر:

$$\begin{aligned} t=۱ &\rightarrow w=۰ \\ t=۲ &\rightarrow w=\frac{\epsilon}{۲} \\ t=۳ &\rightarrow w=\frac{\epsilon}{۲}+\frac{\epsilon}{۲} \\ &\vdots \\ t=t &\rightarrow w=(t-۱)\frac{\epsilon}{۲} \end{aligned} \tag{۳-۷}$$

است. در نتیجه با توجه به رابطه ۳-۷، تعداد کل فازها برابر مقدار ثابت زیر است:

$$\sum\nolimits_{t=۱}^{\frac{۲}{\epsilon}} \frac{۲(\epsilon+w)}{\epsilon} = \sum\nolimits_{t=۱}^{\frac{۲}{\epsilon}} t+۱ = O\left(\left(\frac{۲}{\epsilon}\right)^{۲}\right) \tag{۳-۸}$$

هر فاز نیز که خود شامل فراخوانی رویه $FindLMSubset$ است، به تعداد ثابتی دور ارتباطی احتیاج دارد. در نتیجه کل الگوریتم با تعداد ثابتی دور ارتباطی قابل اجراست. از طرفی فراخوانی رویه $FindLMSubset$ در هر فاز نیاز به زمان ترتیبی $O\left(\frac{n}{p}\log n\right)$ دارد و با توجه به تعداد فازها در رابطه ۳-۸، زمان اجرای کل الگوریتم برابر $O\left(\frac{۱}{\epsilon^{۲}}\frac{n}{p}\log n\right)$ است. □

نتیجه ۱۸.۳.۳ غشای محدب n نقطه در صفحه، با استفاده از یک ماشین $CGM\left(n,p\right)$ و در زمان $O\left(\frac{T_{sequential}}{p}+T_s\left(n,p\right)\right)$ قابل محاسبه است که $T_s\left(n,p\right)$ زمان مرتب‌سازی n داده بر روی یک ماشین موازی با p پردازنده است. علاوه بر این، محاسبه غشا تنها به تعداد ثابتی دور ارتباطی کلی و حافظه $\frac{n}{p}>p^{\epsilon}$ احتیاج دارد که ϵ یک مقدار ثابت کوچک، مثبت و دلخواه است.

www.jozve.org

www.jozve.org

فصل ۴

بررسی کارایی عملی الگوریتم‌ها

www.jozve.org

به منظور بررسی نحوه عملکرد واقعی الگوریتم CGM محاسبه غشای فوقانی و همچنین بررسی کارایی و مقیاس‌پذیری آن، نتایج پیاده‌سازی الگوریتم $MergeHull۱$ با فضای حافظه $\frac{n}{p} \geq p^{۲}$، که توسط دیالو و همکارانش بر روی کامپیوتر چندپردازنده‌ای CRAY T3E [۳] انجام شده است، ارائه می‌گردد. این کامپیوتر ۲۵۶ پردازنده دارد که در پیاده‌سازی انجام شده، تنها از ۶۴ پردازنده آن استفاده شده است. برای این‌که صرفاً رفتار الگوریتم محاسبه غشای محدب بررسی گردد، اکثر آزمایش‌ها بر روی داده‌های مرتب انجام شده است.

برای محاسبه غشای محدب محلی در هر یک از پردازنده‌ها، از الگوریتم ترتیبی [۷] استفاده شده است. این الگوریتم برای داده‌هایی که از قبل مرتب هستند، بسیار مناسب و از مرتبه خطی است و برای یک مجموعه دلخواه از نقاط، دارای پیچیدگی $O\left(n\log n\right)$ است. هر نقطه به صورت یک زوج دوتایی از اعداد ممیز شناور با یک رقم دقت، نشان داده می‌شود.

در لم ۲.۱ مرجع [۱۸] ثابت شده است که اگر S مجموعه‌ای از نقاط در صفحه باشد که این نقاط به طور یکنواخت در صفحه توزیع شده باشند، در این صورت غشای فوقانی S، به طور میانگین تنها شامل چندین نقطه از مجموعه S است. بنابراین در الگوریتم $MergeHull۱$، پس از توزیع یکنواخت نقاط بین پردازنده‌ها و محاسبه هر یک از غشاهای محلی، تعداد نقاط باقی‌مانده بر روی هر یک از غشاهای محلی به مراتب کمتر از $\frac{n}{p}$ خواهد بود. به عبارتی دیگر، مجموع تعداد نقاط باقی‌مانده بر روی تمام غشاهای محلی در ابتدای گام ترکیب، به طور میانگین، خیلی کمتر از تعداد نقاط مجموعه اولیه S است. این کاهش داده می‌تواند به طور قابل ملاحظه‌ای به گام ترکیب موازی سرعت بخشد. زیرا تعداد داده‌هایی که می‌بایست منتقل یابند را کاهش می‌دهد. بنابراین در ادامه فصل رفتار الگوریتم نسبت به بدترین حالت ورودی (یعنی زمانی‌که تمام نقاط مجموعه ورودی در غشای فوقانی ظاهر شوند)، مجموعه‌های داده میانی (یعنی مجموعه‌های داده با تعداد قابل توجهی از نقاط بر روی غشای فوقانی) و مجموعه داده‌های تصادفی مورد بررسی قرار می‌گیرد.

همان‌طور که در بخش ۶.۳.۳ ذکر شد، گام ترکیب الگوریتم $MergeHull۱$، شامل سه مرحله اصلی است:

مرحله ۱: هر پردازنده حداکثر p داده خود را، به عنوان مجموعه جداکننده به پردازنده‌های با شماره بیشتر ارسال می‌کند (در حالتی که تعداد نقاط غشای محدب محلی کمتر از p باشد، کمتر از p داده ارسال می‌گردد). عناصر بعدی مورد نیاز محاسبه شده و بازگردانده می‌شوند.

مرحله ۲: هر پردازنده داده‌های بین دو جداکننده منتخب را به پردازنده‌های با شماره بیشتر ارسال می‌کند. عناصر بعدی مورد نیاز محاسبه شده و بازگردانده می‌شوند.

www.jozve.org

مرحله ۳: هر پردازنده عنصر $Nexts$ خود را به پردازنده‌های با شماره بیشتر ارسال نموده و سپس هر پردازنده بر اساس اطلاعات دریافتی، نقاطی از غشای خود را که می‌بایست در غشای نهایی ظاهر گردند، شناسایی می‌کند.

پردازنده p-ام، پردازنده‌ای است که بیشترین مقدار داده را دریافت می‌کند. این پردازنده، در مرحله ۱ تعداد $p\,(p-1)$، در مرحله ۲، تعداد $\frac{n}{p^2}\,(p-1)$ و در مرحله ۳ تعداد $p-1$ داده را دریافت می‌کند. در هنگام برگشت داده‌ها نیز، حداکثر همین مقدار داده منتقل می‌شود و پردازنده i-ام، نقش پردازنده $p-1-i$ را ایفا می‌کند. بنابراین کل زمان ارتباطات برابر رابطه ۴-۱ است.

$$T_{comm} = O\left(p^2 + \frac{n}{p}\right). \qquad (۱-۴)$$

محاسبات محلی در هر پردازنده، بر روی $\frac{n}{p}$ داده محلی و همچنین داده‌های دریافتی از پردازنده‌های دیگر، انجام می‌شود. در گام اول الگوریتم $MergeHull1$، پیوش نقاط به زمان $O\left(\frac{n}{p}\right)$ احتیاج دارد. در گام دوم الگوریتم، رویه $FindLMSubset$ به تعداد عناصر دریافتی، یعنی p^2 بار اجرا می‌گردد که زمان اجرای آن برابر $p^2 \log \frac{n}{p}$ خواهد بود. در گام چهارم نیز، رویه $FindLMSubset$ بر روی حداکثر $\frac{n}{p}$ داده و با زمان $\frac{n}{p} \log \frac{n}{p}$ اجرا می‌شود. بنابراین زمان محاسبات محلی برابر رابطه ۴-۲ است.

$$T_{loc} = O\left(\frac{n}{p} + p^2 \log \frac{n}{p} + \frac{n}{p} \log \frac{n}{p}\right). \qquad (۲-۴)$$

با توجه به رابطه‌های ۴-۱ و ۴-۲، زمان کل اجرای الگوریتم برابر رابطه ۴-۳ است.

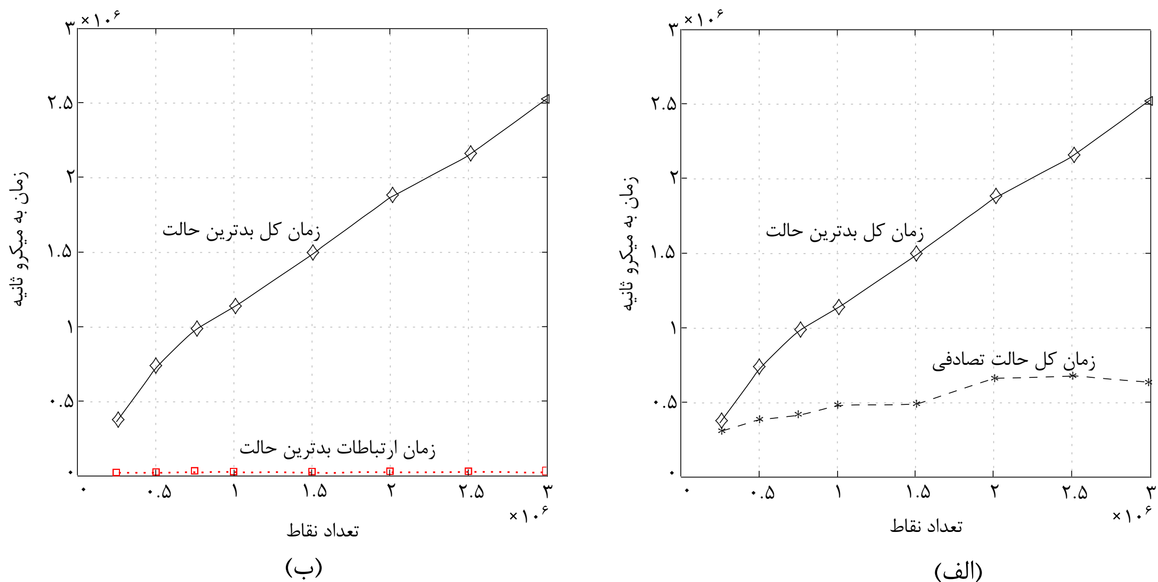
$$T_{tot} = T_{loc} + T_{comm}. \qquad (۳-۴)$$

در ادامه، نتایج آزمایش‌ها آمده است. در کلیه نمودارهای این فصل، محور عمودی نشان‌دهنده زمان اجرا برحسب واحد میکروثانیه است. ترسیم و تحلیل نتایج برای مجموعه نقاط تصادفی، دشوار است زیرا بسته به موقعیت نقاط تصادفی، تعداد نقاط ظاهر شده روی غشای فوقانی متغیر است و روشی برای پیشگویی میزان کاهش تعداد نقاط وجود ندارد. زمان ارتباطات و زمان کل اجرای الگوریتم نیز بسته به تعداد نقاط روی غشاهای محلی، متغیر خواهد بود. بنابراین در رسم، زمان اجرا برای مجموعه نقاط تصادفی، برابر میانگین زمان اجرای الگوریتم بر روی ۱۰ مجموعه نقاط تصادفی، در نظر گرفته شده است. زمان اجرای الگوریتم بر روی بدترین حالت ورودی نیز مورد آزمایش قرار گرفته است. بدترین حالت ورودی حالتی است که همه نقاط ورودی بر روی غشای فوقانی نهایی ظاهر گردند. برای داشتن چنین حالتی، فرض می‌شود که تمام نقاط ورودی روی یک خط راست قرار می‌گیرند.

www.jozve.org

شکل ۱.۴ نتایج آزمایش را در حالتی که تعداد نقاط در حال افزایش و تعداد پردازنده‌ها ثابت است، نشان می‌دهد. شکل ۱.۴ قسمت (الف)، نمودار بدترین حالت ورودی و حالت تصادفی را نشان می‌دهد. دو نمودار با افزایش n نسبت به یکدیگر واگرا هستند. طبق معادله ۴-۲ و ۴-۱، چون p ثابت است، زمان محاسبات محلی متناسب با $n\log n$ و زمان ارتباطات متناسب با n رشد می‌کند. شکل ۱.۴ قسمت (ب)، زمان اجرای کل الگوریتم و زمان ارتباطات را برای بدترین حالت ورودی نشان می‌دهد. زمان محاسبه محلی طبق رابطه ۴-۲ متناسب با $n\log n$ رشد می‌کند. زمان کل ارتباطات بسیار به آرامی رشد می‌کند.

شکل ۲.۴ نتایج آزمایش را در حالتی که تعداد پردازنده‌ها در حال افزایش و تعداد نقاط ثابت است، نشان

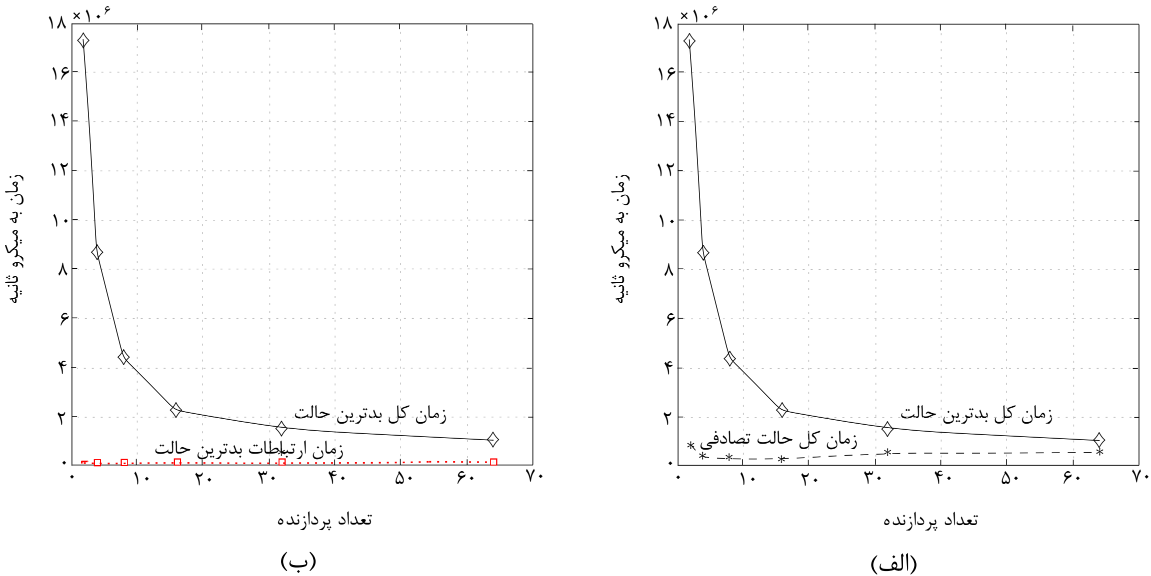


شکل ۱.۴: p ثابت ($p=۳۲$).

می‌دهد. شکل ۲.۴ قسمت (الف)، تسریع را برای حالت تصادفی و بدترین حالت ورودی نشان می‌دهد. در حالت تصادفی، وقتی که تعداد پردازنده‌ها کم است، یک رفتار منظم و رو به کاهش مشاهده می‌شود. اما به تدریج با افزایش تعداد پردازنده‌ها، زمان اجرا نیز افزایش می‌یابد. این افزایش غیر معمول، به‌این علت است که پس از توزیع نقاط تصادفی بین پردازنده‌ها و محاسبه غشاهای محلی در مرحله اول الگوریتم، تعداد نقاط باقی‌مانده بر روی غشاهای محلی طبق لم ۲.۱ مرجع [۱۸]، کاهش می‌یابد و در نتیجه در دو مرحله‌ی بعدی الگوریتم، داده‌های کم می‌بایست بین تعداد زیادی پردازنده منتقل شود که باعث افزایش هزینه ارتباطات می‌گردد. این در حالی است که هزینه محاسبات محلی کم است. از این‌رو، روند کاهش زمان نمودار تا زمانی حفظ می‌شود که زمان برقراری ارتباط نسبت به زمان محاسبه محلی، قابل چشم‌پوشی باشد. در شکل ۲.۴ قسمت (ب)، زمان اجرا به طور یکنواخت کاهش یافته و در نتیجه ضریب تسریع افزایش می‌یابد. این رفتار منظم نمودار به این

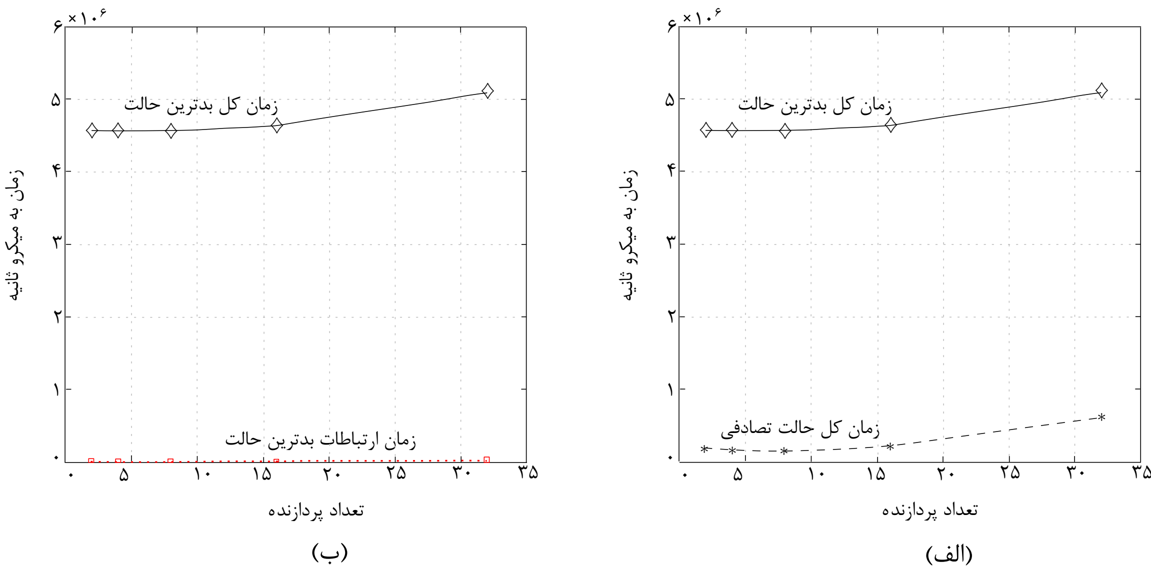
www.jozve.org

علت است که زمان محاسبات محلی (محاسبات غشای محلی و محاسبه عناصر بعدی) به اندازه کافی نسبت به زمان ارتباطات بیشتر است.



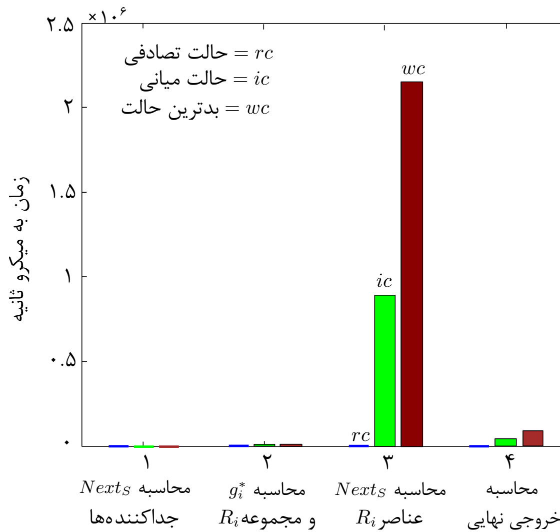
شکل ۲.۴: n ثابت $(n = ۱۵۰۰۰۰۰)$.

شکل ۳.۴ رفتار الگوریتم را در حالتی که نسبت تعداد نقاط به تعداد پردازنده‌ها ثابت و تعداد پردازنده‌ها در حال افزایش است را نشان می‌دهد ($\frac{n}{p} = ۲۰۰۰۰۰$). در هر دو حالت تصادفی و بدترین ورودی، منحنی‌ها شکل یکسانی دارند. برای مقادیر کوچک p مقیاس‌پذیری خوبی مشاهده می‌شود. اما با رشد p، ضریب $p^۲$ به‌تدریج ظاهر می‌گردد.

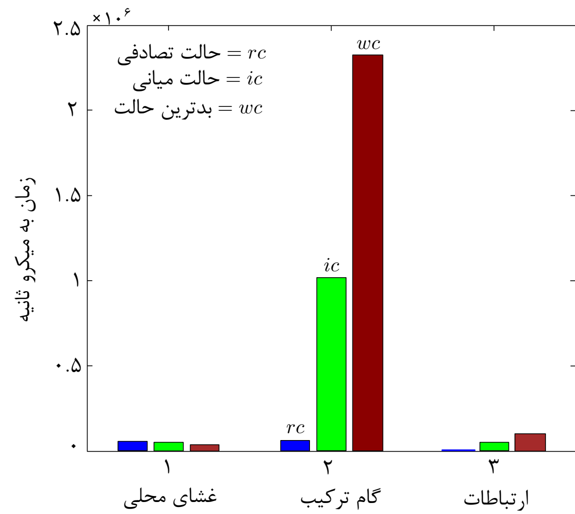


شکل ۳.۴: نسبت $\frac{n}{p}$ ثابت $\left(\frac{n}{p} = ۲۰۰۰۰۰\right)$.

www.jozve.org



(ب)



(الف)

شکل ۴.۴: زمان اجرا به تفکیک گام‌های مختلف الگوریتم ($p = ۸۰, n = ۸۰۰۰۰۰$).

شکل ۴.۴ قسمت (الف)، زمان اجرای الگوریتم را به تفکیک گام‌های مختلف آن و برای سه مجموعه داده تصادفی، میانی و بدترین ورودی با مقادیر $p = ۸$ و $n = ۸۰۰۰۰۰$ نشان می‌دهد. گروه اول میله‌های عمودی، متناظر با زمان محاسبه غشای محلی است. گروه دوم، زمان کل محاسبات گام ترکیب و گروه سوم، زمان کل ارتباطات را نشان می‌دهد.

شکل ۴.۴ قسمت (ب)، زمان محاسبات گام ترکیب را به تفکیک نشان می‌دهد. گروه اول میله‌های عمودی، متناسب با زمان محاسبه عنصر $Nexts$ هر یک از جداکننده‌ها (رجوع کنید به بخش ۳.۳.۳) است. گروه دوم زمان محاسبه سمت چپ‌ترین نقاط و نقاطی که بین جداکننده‌ها قرار می‌گیرند (یعنی مجموعه R_i) را نشان می‌دهد. گروه سوم زمان محاسبه عنصر $Nexts$ هر یک از نقاط مجموعه R_i و گروه آخر زمان محاسبه نقاطی که باید بر روی غشای نهایی ظاهر گردند را نشان می‌دهد.

همانطور که در شکل ۴.۴ قسمت (ب) می‌بینید، بیشترین زمان محاسبه گام ترکیب مربوط به قسمت محاسبه عنصر بعدی نقاطی است که بین دو جداکننده قرار می‌گیرند.

www.jozve.org

فصل ۵

نتایج و پیشنهادات

www.jozve.org

هدف از این پایان‌نامه، مطالعه‌ی ملزومات برنامه‌نویسی موازی و آشنایی با برخی مدل‌های برنامه‌نویسی موازی بود. به علت رشد روزافزون استفاده از کامپیوتر در تمامی عرصه‌های علمی و افزایش نیاز به پردازش‌های سریع‌تر، مطالعه‌ی نحوه برنامه‌نویسی موازی و آشنایی با روش‌های به‌کارگیری سخت‌افزارهای مختلف برای حل مسائل علمی، از اهمیت فراوانی برخوردار است. به همین منظور مسئله هندسی محاسبه غشای محدب n نقطه در صفحه، به عنوان نمونه انتخاب شد و نحوه حل این مسئله با استفاده از یک الگوریتم موازی، مورد بررسی قرار گرفت.

مدلی که برای حل مسئله محاسبه غشای محدب انتخاب گردید، مدل **چندکامپیوتری دانه‌درشت** CGM بود. در این مدل با به‌کارگیری p پردازنده و توزیع مجموعه نقاط بین این پردازنده‌ها، تلاش می‌شود که مسئله به صورت موازی حل گردد و در مواردی که نیاز به برقراری ارتباط بین پردازنده باشد، این نیاز به وسیله انتقال پیام بین پردازنده‌ها مرتفع می‌گردد. ویژگی بارز این مدل که آن را از مدل‌های موازی مشابه متمایز می‌سازد این است که کلیه ارتباطات مورد نیاز همزمان و توسط یک رابطه h-تایی قابل انجام است و در نتیجه هزینه ارتباطات تنها با تعداد دورهای ارتباطی قابل سنجش است.

الگوریتم موازی انتخاب شده برای محاسبه غشای محدب در یک ماشین $CGM\left(n,p\right)$، ابتدا مجموعه نقاط ورودی را بین p پردازنده موجود به طور یکنواخت توزیع کرده و سپس غشاهای محلی را در زمان $O\left(\frac{n\log n}{p}\right)$ محاسبه می‌کند.

الگوریتم $MergeHull۱$ با استفاده از رویه‌های $QueryFindNext$ و $FindLMSubset$، ترکیب p غشای محلی را در زمان $O\left(\frac{n\log n}{p}+T_s\left(n,p\right)\right)$ محاسبه می‌کند که $T_s\left(n,p\right)$ زمان مرتب‌سازی n داده بر روی یک ماشین موازی با p پردازنده است. بنابراین زمان اجرای کل الگوریتم برابر $O\left(\frac{n\log n}{p}+T_s\left(n,p\right)\right)$ است که در صورت بهینه بودن الگوریتم مرتب‌سازی به کار رفته، الگوریتم محاسبه غشای محدب بهینه خواهد بود. زمان حل مسئله غشای محدب به روش ترتیبی برابر $\Omega\left(n\log n\right)$ است. در نتیجه با به کارگیری روش موازی، ضریب تسریع برابر p خواهد بود.

الگوریتم ترکیب $MergeHull۱$ تنها به ۵ دور ارتباطی احتیاج دارد اما از لحاظ فاکتور مقیاس‌پذیری بهینه نیست زیرا تنها برای نسبت‌های $\frac{n}{p}>p^۲$ قابل اجراست.

الگوریتم $MergeHull۲$، تعداد p غشای فوقانی را که هر یک بر روی یک پردازنده ذخیره شده‌اند را به عنوان ورودی دریافت می‌کند. سپس با تقسیم کل پردازنده‌ها به $p^{\frac{۴}{۴}}$ گروه، هر گروه شامل $p^{\frac{۴}{۴}}$ پردازنده و با به‌کارگیری رویه‌های $BuildHulls$، $FindLMSubset$ و $QueryFindNext$، ترکیب غشاهای محلی را در تعداد

www.jozve.org

ثابتی فاز محاسبه می‌کند. مزیت این الگوریتم نسبت به الگوریتم $MergeHull1$ در مقیاس‌پذیری آن برای نسبت‌های $p^{\epsilon} > \frac{n}{p}$ است که ϵ یک مقدار ثابت کوچک، مثبت و دلخواه است. علاوه بر این، همچنان الگوریتم به تعداد ثابتی دور ارتباطی و زمان $O\left(\frac{n\log n}{p}+T_s\left(n,p\right)\right)$ احتیاج دارد. نتایج آزمایش‌ها نشان می‌دهد که بیشترین زمان پیاده‌سازی الگوریتم محاسبه غشای فوقانی مجموعه S، مربوط به گام ترکیب که شامل فراخوانی الگوریتم $MergeHull1$ است، می‌باشد و بیشترین زمان محاسبه گام ترکیب نیز مربوط به قسمت محاسبه عنصر بعدی نقاطی است که بین دو جداکننده قرار می‌گیرند. بنابراین به نظر می‌رسد که این بخش از محاسبات، تنگنای الگوریتم ارائه شده است و در پیاده‌سازی، بیشترین زمان را به خود اختصاص می‌دهد. بنابراین بهبود این گام، به عنوان مسئله‌ای برای کارهای تحقیقاتی آینده باقی می‌ماند. از طرفی به نظر می‌رسد پیاده‌سازی این الگوریتم بر روی سیستم‌های چندهسته‌ای به علت کمتر بودن زمان‌پاسخگویی این سیستم‌ها نسبت به سیستم‌های چندپردازنده‌ای، دارای نتایج بهتری باشد. بنابراین سعی می‌شود تا در کارهای تحقیقاتی آینده، این الگوریتم بر روی یک کامپیوتر چندهسته‌ای پیاده‌سازی گردد و نتایج به‌دست آمده با نتایج فعلی مقایسه گردد.

www.jozve.org

واژه‌نامه فارسی به انگلیسی

آرایه پردازنده‌ها ... Array processers

آرایه سیستولیک ... Systolic Array

ابرگام ... Superstep

انتشار ... Broadcast

انتشار بسته ... Segment broadcast

انتشار بسته یک‌تایی ... Segment 1-broadcast

انتشار کلی ... All-to-All broadcast

بارمفید ... Payload

بسته ... Packet

پراکندن ... Scatter

پردازه ... Process

پسین ... Successor

پیشین ... Predecessor

تبادل کلی ... Total exchange

تجمیع بسته ... Segment gather

تراشه چندپردازنده ... Chip multiprocessor

ترافلاپس ... TeraFLOPS

ترکیب ... Combine

تسریع ... Speed Up

تک‌باس ... Single Bus

تک‌نخی ... Single-threading

تنگناها ... Bottlenecks

جداکننده‌ها ... Splitters

جمع جزئی ... Partial sum

جمعی ... Collective

چندباس ... Multiple Bus

چندپردازنده متقارن ... Symmetric Multiprocessor (SMP)

چندکامپیوترها ... Multicomputers

چندکامپیوتری دانه‌درشت ... Coarse Grained Multicomputers

چندلایه‌ای ... Multistage

چندنخی ... Multithreading

چندنخی همزمان ... Simultaneous multithreading

حافظه توزیع شده ... Distributed Memory

www.jozve.org

حافظه مشترک ... Shared Memory

خواندن انحصاری، نوشتن انحصاری ... Exclusive-Read Exclusive-Write

خواندن انحصاری، نوشتن همزمان ... Exclusive-Read Concurrent-Write

خواندن همزمان، نوشتن انحصاری ... Concurrent -Read Exclusive -Write

خواندن همزمان، نوشتن همزمان ... Concurrent -Read Concurrent -Write

دانه‌بندی ... Granularity

دانه‌بندی درشت ... Coarse Granularity

دانه‌بندی ریز ... Fine Granularity

دانه‌درشت ... Coarse Grained

دانه‌ریز ... Fine Grained

رابطه h-تایی ... h-relation

سرآیند ... Header

سیستم‌های اطلاعاتی جغرافیایی ... Geographic Information System (GIS)

شبکه‌های ارتباطی سیستم‌های چندپردازنده ... Multiprocessors Interconnection Networks

طبقه‌بندی فلین ... Flynn’s Taxonomy

عنصر پردازشگر ... Processing Element

غشای تحتانی ... Lower hull

غشای فوقانی ... Upper hull

غشای محدب ... Convex hull

فناوری واقعیت مجازی ... Virtual Reality Technology

فوق مکعبی ... Hypercube

قانون آمدال ... Amdahl’s law

قطعه نازک مستطیل‌شکل ... Die

قلم داده ... Data item

کارایی ... Efficiency

کامپیوترهای چندهسته‌ای ... Multi core computers

کامپیوترهای موازی خوشه‌ای ... Cluster parallel computers

کراس‌بار ... Crossbar

گردآوری ... Gather

ماشین دسترسی تصادفی موازی ... Parallel Random Access Machine

مافوق نخ ... Hyper-Threading

مانع همگام‌سازی ... Barrier

محدب ... Convex

مرتب‌سازی کلی ... Global sort

www.jozve.org

مسیریاب .... Router

مش .... Mesh

مقیاس‌پذیری .... Scalability

نخ .... Thread

نقاط اصلی .... Hotspots

نقطه به نقطه .... Point-to-Point

وابستگی .... Coherency

واحد پردازش داده .... Data Processing Unit

واحد پردازش مرکزی .... Central Processing Unit

واحد حساب و منطق .... Arithmetic Logic Unit

واحد کنترل .... Control unit

ورودی-خروجی .... Input-Output

وظیفه .... Task

هسته .... Core

همروند .... Concurrent

www.jozve.org

واژه‌نامه انگلیسی به فارسی

All-to-All broadcast .... انتشار کلی
Amdahl's law .... قانون آمدال
Arithmetic Logic Unit .... واحد حساب و منطق
Array processers .... آرایه پردازنده‌ها
Barrier .... مانع همگام‌سازی
Bottlenecks .... تنگناها
Broadcast .... انتشار
Central Processing Unit .... واحد پردازش‌مرکزی
Chip multiprocessor .... تراشه چندپردازنده
Cluster parallel computers .... کامپیوترهای موازی خوشه‌ای
Coarse Grained .... دانه‌درشت
Coarse Grained Multicomputers .... چندکامپیوتری دانه‌درشت
Coarse Granularity .... دانه‌بندی درشت
Coherency .... وابستگی
Collective .... جمعی
Combine .... ترکیب
Concurrent .... همروند
Concurrent -Read Concurrent -Write .... خواندن همزمان، نوشتن همزمان
Concurrent -Read Exclusive -Write .... خواندن همزمان، نوشتن انحصاری
Control unit .... واحد کنترل
Convex .... محدب
Convex hull .... غشای محدب
Core .... هسته
Crossbar .... کراس‌بار
Data item .... قلم داده
Data Processing Unit .... واحد پردازش داده
Die .... قطعه نازک مستطیل‌شکل
Distributed Memory .... حافظه توزیع شده
Efficiency .... کارایی
Exclusive-Read Concurrent-Write .... خواندن انحصاری، نوشتن همزمان
Exclusive-Read Exclusive-Write .... خواندن انحصاری، نوشتن انحصاری
Fine Grained .... دانه‌ریز
Fine Granularity .... دانه‌بندی ریز

www.jozve.org

Flynn’s Taxonomy . . . طبقه‌بندی فلین

Gather . . . گردآوری

Geographic Information System (GIS) . . . سیستم‌های اطلاعاتی جغرافیایی

Global sort . . . مرتب‌سازی کلی

Granularity . . . دانه‌بندی

Header . . . سرآیند

Hotspots . . . نقاط اصلی

h-relation . . . رابطه h-تایی

Hypercube . . . فوق مکعبی

Hyper-Threading . . . مافوق نخ

Input-Output . . . ورودی–خروجی

Lower hull . . . غشای تحتانی

Mesh . . . مش

Multi core computers . . . کامپیوترهای چندهسته‌ای

Multicomputers . . . چندکامپیوترها

Multiple Bus . . . چندباس

Multiprocessors Interconnection Networks . . . شبکه‌های ارتباطی سیستم‌های چندپردازنده

Multistage . . . چندلایه‌ای

Multithreading . . . چندنخی

Packet . . . بسته

Parallel Random Access Machine . . . ماشین دسترسی تصادفی موازی

Partial sum . . . جمع جزئی

Payload . . . بارمفید

Point-to-Point . . . نقطه به نقطه

Predecessor . . . پیشین

Process . . . پردازه

Processing Element . . . عنصر پردازشگر

Router . . . مسیریاب

Scalability . . . مقیاس‌پذیری

Scatter . . . پراکندن

Segment 1-broadcast . . . انتشار بسته یک‌تایی

Segment broadcast . . . انتشار بسته

Segment gather . . . تجمیع بسته

Shared Memory . . . حافظه مشترک

www.jozve.org

Simultaneous multithreading . . . . چندنخی همزمان
Single Bus . . . . تک‌باس
Single-threading . . . . تک‌نخی
Speed Up . . . . تسریع
Splitters . . . . جداکننده‌ها
Successor . . . . پسین
Superstep . . . . ابرگام
Symmetric Multiprocessor (SMP) . . . . چندپردازنده متقارن
Systolic Array . . . . آرایه سیستولیک
Task . . . . وظیفه
TeraFLOPS . . . . ترافلاپس
Thread . . . . نخ
Total exchange . . . . تبادل کلی
Upper hull . . . . غشای فوقانی
Virtual Reality Technology . . . . فناوری واقعیت مجازی

www.jozve.org

مراجع

[1] S. Akhter and J. Roberts. *Multi-Core Programming: Increasing Performance through Software Multithreading*. Intel Press, 2006.

[2] S. G. Akl. A constant-time parallel algorithm for computing convex hulls. *Bit Numerical Mathematics*, 22:130–134, 1982.

[3] E. Anderson, J. Brooks, C. Grassl, and S. Scott. Performance of the CRAY T3E multiprocessor. In *Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–17, 1997.

[4] M. J. Atallah and J. J. Tsay. On the parallel decomposability of geometric problems. In *Proceedings of the 5th annual symposium on Computational geometry*, SCG '89, pages 104–113, New York, NY, USA, 1989. ACM.

[5] D. Avis. On the complexity of finding the convex hull of a set of points. *Discrete Applied Mathematics*, 4:780–787, 1982.

[6] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, AFIPS '68 (Spring), pages 307–314. ACM, 1968.

[7] J. L. Bentley, K. L. Clarkson, and D. B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. *Algorithmica*, pages 168–183, 1993.

[8] L. Boxer and R. Miller. *Algorithms Sequential & Parallel: A Unified Approach*. Charles River Media, second edition, 2005.

[9] F. P. Brooks. What's real about virtual reality? *IEEE Computer Graphics and Applications*, 19(6):16–27, 1999.

www.jozve.org

[10] A. Chan and F. Dehne. CGMgraph/CGMlib: Implementing and Testing CGM Graph Algorithms on PC Clusters. In *Proceedings of the 10th Parallel Virtual Machine and Message Passing Interface conference*, Lecture Notes in Computer Science, pages 117–125. Springer, 2003.

[11] A. Chow. A parallel algorithm for determining convex hulls of sets of points in two dimensions. In *Annual Allerton Conference on Communication, Control and Computing*, volume 19, pages 214–223, 1981.

[12] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPOPP '93, pages 1–12, New York, NY, USA, 1993. ACM.

[13] M. de Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition, 2008.

[14] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. A. Khokhar. A randomized parallel 3D convex hull algorithm for coarse grained multicomputers. In *Proceedings of the 7th annual ACM symposium on Parallel algorithms and architectures*, SPAA '95, pages 27–33, New York, NY, USA, 1995. ACM.

[15] F. Dehne, A. Fabri, and A. Rau-chaplin. Scalable parallel computational geometry for coarse grained multicomputers. *International Journal on Computational Geometry*, 6:298–307, 1994.

[16] F. Dehne, A. Ferreira, E. Caceres, S. W. Song, and A. Roncato. Efficient parallel graph algorithms for coarse-grained multicomputers and BSP. *Algorithmica*, pages 183–200, 2002.

[17] F. Dehne, C. Kenyon, and A. Fabri. Scalable and architecture independent parallel geometric algorithms with high probability optimal time. In *Proceedings of the 6th annual IEEE symposium on Parallel and Distributed Processing*, pages 586 – 593. IEEE Comp. Soc. Press, 1994.

[18] L. Devroye and G. T. Toussaint. A note on linear expected time algorithms for finding convex hulls. *Computing*, 26(4):361–366, 1981.

www.jozve.org

[19] M. Diallo, A. Ferreira, A. Rau-Chaplin, and S. Ubéda. Scalable 2D Convex Hull and Triangulation Algorithms for Coarse Grained Multicomputers. *parallel and distributed computing*, 56(1):47–70, 1999.

[20] H. El-Rewini and M. Abd-El-Barr. *Advanced Computer Architecture and Parallel Processing*. Wiley-Interscience, 2005.

[21] M. T. Goodrich. Communication-efficient parallel sorting. *SIAM Journal on Computing*, 29(2):416–432, 1999.

[22] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *Proceedings of the 1993 IEEE 34th Annual Foundations of Computer Science*, pages 714–723, Washington, DC, USA, 1993. IEEE Computer Society.

[23] A. Grama, G. Karypis, V. Kumar, and A. Gupta. *Introduction to Parallel Computing*. Addison Wesley, second edition, 2003.

[24] S. E. Hambrusch and A. A. Khokhar. C3: An architecture-independent model for coarse-grained parallel machine. In *Proceedings of the 6th annual IEEE symposium on Parallel and Distributed Processing*, pages 544 – 551, 1994.

[25] M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann, 2008.

[26] H. Li and K. C. Sevcik. Parallel sorting by over partitioning. In *Proceedings of the 6th annual ACM symposium on Parallel algorithms and architectures*, SPAA '94, pages 46–56, New York, NY, USA, 1994. ACM.

[27] J. M. Marberg and E. Gafni. Sorting in constant number of row and column phases on a mesh. *Algorithmica*, 3:561–572, 1988.

[28] R. Miller and Q. F. Stout. Computational geometry on a mesh-connected computer. In *International Conference on Parallel Processing*, volume 1984, pages 236–271, 1984.

[29] R. Miller and Q. F. Stout. Efficient parallel convex hull algorithms. *IEEE Transactions on Computers*, 37:1605–1618, 1988.

[30] B. Parhami. *Introduction to Parallel Processing: Algorithms and Architectures*. Kluwer Academic Publishers, 2002.

www.jozve.org

[31] F. P. Preparata and M. I. Shamos. *Computational geometry: an introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[32] E. L. G. Saukas and S. W. Song. A parallel algorithm for solving tridiagonal linear systems on coarse grained multicomputer. In *IX Brazilian symposium on Computer Architecture and High-Performance Processing*, pages 463 – 474, 1997.

[33] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. Wiley Publishing, eighth edition, 2008.

[34] W. Stallings. *Operating Systems: Internals and Design Principles(GOAL Series)*. Prentice Hall, sixth edition, 2008.

[35] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[36] U. Vishkin, G. C. Caragea, and B. C. Lee. *Handbook of Parallel Computing: Models, Algorithms and Applications*, chapter Transitional Issues Fine-Grain to Coarse-Grain Multicomputers. CRC Press, 2007.

[37] D. X. and G. N. Good algorithm design style for multiprocessors. In *The 6th IEEE Symposium on Parallel and Distributed Processing*, pages 538–543, 1994.

[38] A. C. C. Yao. A lower bound to finding convex hulls. *Journal of the ACM(JACM)*, 28(4):780–787, 1981.

www.jozve.org

Abstract

The convex hull of a set of points is the smallest convex set that contains the points. The convex hull is a fundamental construction for mathematics and computational geometry and it is applied widely in pattern recognition, morphology and image processing.

In this thesis, a scalable parallel algorithm for building the convex hull of a set of n points in the plane is studied. This algorithm is designed for the coarse grained multicomputer model: p processors with $O\left(\frac{n}{p}\right) \gg O(1)$ local memory each, connected to some arbitrary interconnection network. They scale over a large range of values of n and p, assuming only that $\frac{n}{p} \geq p^{\epsilon}\,(\epsilon > 0)$ and require time $\left(\frac{T_{sequential}}{p} + T_s(n,p)\right)$, where $T_{sequential}$ refers to the time of a best sequential algorithm and $T_s(n,p)$ refers to the time of a global sort of n data on a p processors machine. Furthermore, they involve only a constant number of global communication rounds.

www.jozve.org

www.jozve.org

Yazd University

Faculty of Mathematics

Department of Computer Science

Thesis submitted

for the degree of Master of Science

Title:

Parallel geometric algorithms for multi-core computers

Supervisor:

Dr. Mohammad Farshi

Advisors:

Dr. Seyed Abolfazl Shahzadeh-Fazeli

Dr. Fazlollah Adibnia

By:

Fatemeh Dehghani Firozabadi

February, 2012

www.jozve.org