موازی سازی الگوریتم DeWall برای حل مثلث بندی دلانی

امیرعلی خسروی دهکردی[۱] ، محمد قدسی[۲]
دانشگاه صنعتی شریف، دانشکده مهندسی کامپیوتر
A_khosravi@ce.sharif.edu

چکیده

از مسائل کلاسیک هندسه محاسباتی مثلث بندی دلانی میباشد که الگوریتم های مختلفی برای آن ارائه شده است یکی از این الگوریتم ها الگوریتم DeWall است که از نوعی روش تقسیم و حل استفاده مینماید، ویژگی اصلی این الگوریتم در اینست که میتوان آنرا بسادگی به ابعاد بالاتر بسط داد، دراین مقاله راهکاری برای موازی سازی الگوریتم DeWall که برای حل مسئله مثلث بندی دلانی بکار میرود ارائه خواهیم کرد. این راهکار با استفاده از تقسیم نقاط صفحه به زیربلوک های مختلف و مثلث بندی آنها بصورت جداگانه و موازی با استفاده از الگوریتم فوق میباشد و علاوه برسادگی دارای بازدهی نسبتا بالا میباشد. نتایج حاصل از اجرای الگوریتم برروی تعداد زیاد نقاط و با توزیع یکنواخت قابل قبول میباشد.

واژه های کلیدی: پردازش موازی - مثلث بندی دلانی - الگوریتم DeWall

۱- مقدمه

یکی از مسائل قدیمی و کلاسیک هندسه محاسباتی مثلث بندی دلانی میباشد که روش های مختلفی توسط محققان برای آن ارائه گردیده است. مثلث بندی دلانی، نوعی مثلث بندی است که در آن دایره محیطی هر مثلث شامل هیچ راسی از رئوس مثلث های دیگر نباشد (شرط دایره محیطی) و واضح است که این نوع مثلث بندی یکتا خواهد بود. راه حلهای مختلف، برای مثلث بندی دلانی بصورت مستقیم عموما در دسته های زیر قرار میگیرند :

- روش بهینه سازی محلی[۳] : به اینصورت میباشد که ابتدا یک مثلث بندی دلخواه برروی نقاط انجام میگیرد و بعد درجهت برقرار بودن شرط دایره تهی محل یالهایی که غیر مجاز هستند تغییر داده میشوند.
- روش برخط [۴]: در این نوع روشها در هر لحظه یک نقطه به مجموعه نقاط افزوده میگردد، این نقطه جدید احتمالا در یکی از چندضلعی های ساخته شده قبلی قرار خواهد گرفت، چند ضلعی که آنرا دربرگرفته است، تقسیم شده و چند، چند ضلعی جدید بوجود می آید، شرط دایره تهی برای هر چند ضلعی جدید بدست آمده کنترل میگردد.

---

[۱] دانشجوی کارشناسی ارشد دانشکده مهندسی کامپوتر، داشنگاه صنعتی شریف
[۲] استاد دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف

[۳] local improvement
[۴] online

- روش افزایشی[1] : در این نوع روشها از یک چند ضلعی شروع کرده، در هر مرحله برروی یکی از یالهای چند ضلعی قبلی یک چند ضلعی جدید اضافه میکنیم، به اینصورت که چند ضلعی جدید شرط دایره تهی را ارضاء نماید.
- روش تعبیه با ابعاد بیشتر[2] : این الگوریتم ها نقاط را به فضای E^{d+1} منتقل میکنند و سپس پوش محدب[3] نقاط را محاسبه کرده و مثلث بندی دلانی را با استفاده از تصویر این نقاط محاسبه شده برروی E^d بدست می آورند ، الگوریتم بلوچی و ... دسته ای از این نوع الگوریتم ها میباشد.
- روش تقسیم و تکرار[4] : این الگوریتمها برگرفته از روش تقسیم و حل عادی میباشند، به اینصورت که مجموعه نقاط به دسته های کوچکتر تقسیم شده و مثلث بندی برای هر یک از این دسته ها انجام میشود و سپس راه حلها بایکدیگر ترکیب شده و مثلث بندی نهایی حاصل میشود، تنها مشکل این روش ترکیب راه حلها میباشد چراکه درزمان ترکیب ممکن است مثلث بندی مرزها تغییر نماید.

در ادامه در بخش۲ خلاصه ای از تحقیقات انجام شده برروی موازی سازی مثلث بندی دلانی را مرور خواهیم کرد، در قسمت ۳ الگوریتم موازی پیشنهاد شده معرفی میگردد، در قسمت ۴ در رابطه با پیاده سازی الگوریتم توضیحاتی ارائه خواهد شد و نهایتا در قسمت ۵ نتیجه گیری قرار داده شده است.

۲-کارهای مرتبط

بطورکلی مسائلی مانند مثلث بندی که دارای حجم بالای داده ورودی میباشند، جهت پردازش موازی مناسبتر میباشند . الگوریتمهای موازی مختلفی برای حل مسئله مثلث بندی دلانی پیشنهاد شده اند که دراین بین الگوریتمهای مبتنی برتقسیم وحل در اجرای موازی دارای بازدهی کمی میباشند، در این بین الگوریتم chen [8] و [13] به بازدهی معادل با ۳ برروی ۸ پردازنده دست یافته است، الگوریتم cignoni و همچنین الگوریتم Hardwick که برای مثلث بندی دلانی ۲ بعدی استفاده شده است الگوریتم های موازی دیگری میباشند که برای مثلث بندی دلانی بکار رفته‌اند، روشی که در [8] و [13] توسط Chen ارائه و پیاده سازی شده است، برمبنای ساخت دیواره با استفاده از الگوریتم افزایشی و مثلث بندی زیر بلوکها بر اساس الگوریتم Dwyer میباشد. هر چند که این الگوریتم بنابرادعای نویسندگان مقاله، الگوریتم بهینه میباشد ولی دارای مشکلاتی دررابطه با ترکیب واسط بین زیربلوکها و بلوکها خواهد بود، علاوه بر آنکه این روش مثلث بندی دلانی تقریبی را ارائه می کند، بهنگام ترکیب باید تعدادی از مثلثهای غیردلانی در زیربلوکها حذف شوند که به این منظور نیز نویسنده مقاله الگوریتمی ارائه نموده است.

درمقابل روش ارائه شده در این گزارش نیازمند تشخیص مثلثهای غیردلانی در زیربلوک نمیباشد و از این جهت نسبت به الگوریتم ارائه شده ساده تر خواهد بود، ضمن آنکه هنگامی که تعداد نقاط تولید شده زیاد باشند، بطوریکه ۴ نقطه برروی یک دایره قرار گیرند، الگوریتم مثلث بندی دلانی ارائه شده در [8] و [13] ممکن است با مشکلاتی در رابطه با ترکیب واسط و زیربلوک مواجه شود.

در این گزارش سعی براینست که الگوریتمی مناسب برای مثلث بندی دلانی ارائه دهیم، این الگوریتم با استفاده از تقسیم نقاط صفحه به بلوکهای مختلف و مثلث بندی آنها بصورت جداگانه با استفاده از الگوریتم ارائه شده DeWall انجام میگردد، ویژگی اصلی این الگوریتم سادگی آن و قابلیت بسط آن به محیط با ابعاد بیشتر خواهد بود، علاوه بر آن در این الگوریتم هر پردازنده برای مثلث بندی ناحیه مربوط به خود تنها نیازمند تبادل داده با پردازنده اصلی و یک یا حداکثر دو پردازنده همسایه خود خواهد بود.

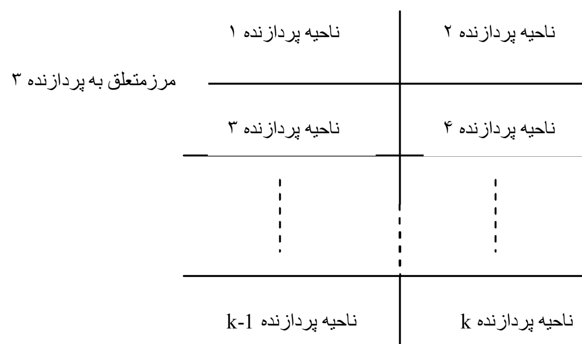
---
[1] incremental
[2] higher dimensional embedding
[3] Convex Hull
[4] Devide and Conquer

۳ - الگوریتم موازی ارائه شده

برخلاف آنچه قبلا توسط طراحان الگوریتم DeWall برای موازی سازی الگوریتم پیشنهاد شده است که برپایه تقسیم وحل و استفاده از ایده اختصاص پردازنده بصورت پویا میباشد، این الگوریتم براساس تقسیم نقاط به بلوکها عمل مینماید، به اینترتیب که ابتدا پردازنده اصلی بکمک پردازنده های دیگر، ساخت دیواره ها را بصورت افزایشی با استفاده از الگوریتم DeWall انجام میدهد، سپس دیواره های مربوط به هر پردازنده توسط پردازنده اصلی و پردازنده های همسایه به آن ارسال میشوند و نقاط نیز به دسته هایی با اندازه مناسب تقسیم شده و بین پردازنده ها توزیع میگردند پردازنده دریافت کننده دیواره ها با استفاده از الگوریتم DeWall مثلث بندی را در محدوده نقاط متعلق به خود انجام میدهد و حاصل مثلث بندی را به پردازنده اصلی ارسال مینماید، برای فهم آسانتر، الگوریتم درفضای دو بعدی شرح داده میشود.

در حقیقت مراحل انجام الگوریتم بصورت زیر خواهد بود :

۱- پردازنده اصلی، صفحه نقاط را به نواحی مختلف براساس تعداد پردازنده ها تقسیم میکند، در ادامه شرح داده میشود که این تقسیم نقاط برای بالابردن بازدهی میتواند بصورت یکسان نباشد. مناطق صفحه به اینصورت تقسیم میشوند که یک خط افقی (یا عمودی) صفحه را به دونیم تقسیم میکند و مجددا هریک از دو نیم صفحه به تعداد نصف پردازنده های باقیمانده به مناطق مختلف تقسیم میگردند . هر دیواره ساخته شده متعلق به پردازنده سمت راست دیواره میباشد .(شکل ۱)



شکل ۱. بلوک بندی نقاط صفحه با استفاده از خط عمودی

۲- پردازنده اصلی ساخت دیواره را برروی خط افقی(عمودی) آغاز مینماید به اینصورت که نزدیکترین نقطه به خط یعنی P_1 را انتخاب مینماید، نزدیکترین نقطه به P_1 را که درجهت دیگر خط باشد انتخاب میکند و برای انتخاب راس سوم از تابع dd معرفی شده در الگوریتم DeWall استفاده میگردد و درحقیقت راس P_3 باید این تابع را کمینه نماید.

۳- پردازنده اصلی با استفاده از الگوریتم ارائه شده و با داشتن یالهای مثلث ساخته شده و لیست فعال آن و با درنظر گرفتن مقدار تابع dd مثلث های دیگر را بصورت افزایشی میسازد تا اینکه به خط مرزی یکی از مناطق محدوده برسد . وقتی یکی از یالهای مثلث ساخته شده دیواره مربوط به یکی از مناطق را قطع کرد یال قطع کننده به پردازنده صاحب دیواره فرستاده میشود.

۴- پردازنده صاحب دیواره به محض دریافت یال ساخت دیواره مربوطه را با استفاده از روش افزایشی آغاز میکند و درهمین حال پردازنده اصلی ساخت دیواره را ادامه میدهد.

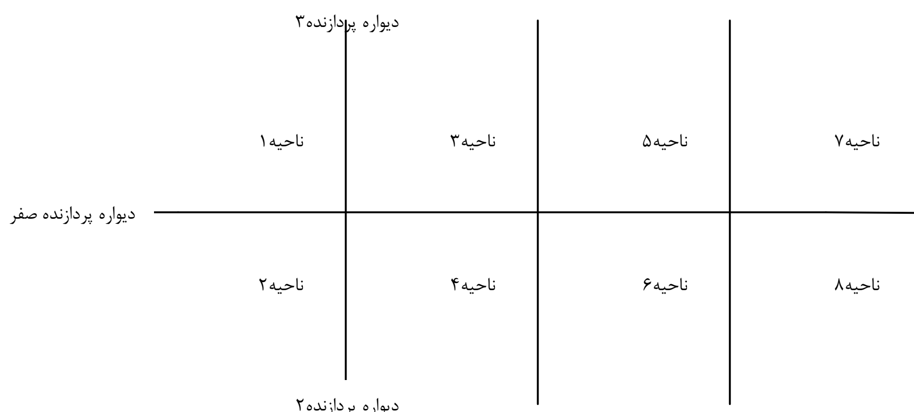
۵- هرپردازنده ای بعد از ساخت دیواره مربوط به خود یالهای سمت چپ دیواره (AFL^{-}) را به پردازنده سمت چپ خود ارسال میکند و منتظر دریافت یالهای دیواره از پردازنده سمت راست خود و نیز از پردازنده اصلی میشود.

۶- پردازنده اصلی بعد از اتمام دیواره مربوط به ناحیه هر پردازنده آنرا به پردازنده مربوطه ارسال میکند.

۷- هر پردازنده بعد از آنکه دیواره را از پردازنده سمت راست و پردازنده اصلی دریافت نمود بهمراه دیواره AFL^{+} که خود قبلا ساخته است لیست فعال AFL را ساخته و تابع اصلی DeWall را برروی آن اجرا مینماید.

۸- بعد از آنکه هر پردازنده ای مثلث بندی دلانی را درناحیه مربوط به خود انجام داد ، آنرا به پردازنده اصلی ارسال مینماید.

۹- پردازنده اصلی با دریافت مثلث بندی از پردازنده های مختلف و دیواره ساخته شده توسط خود، مثلث بندی نهایی را بدست میدهد.



شکل ۲. دیواره های ساخته شده و ناحیه های متعلق به هر پردازنده

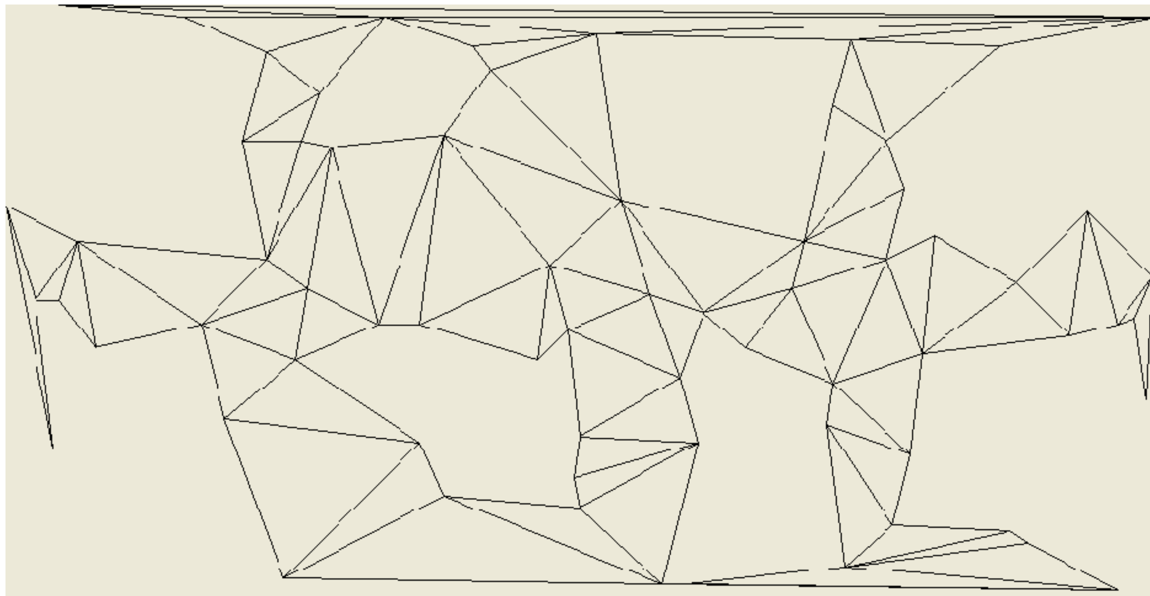
۳-۱- جزییات الگوریتم ارائه شده

اختصاص پردازنده بصورت پویا موجب استفاده بهینه از قدرت پردازنده ها نخواهد گشت، چراکه در مراحل اولیه الگوریتم از تعدادی از پردازنده ها استفاده نمیشود، بهمین منظور و به جهت استفاده بهتر از پردازنده ها میتوانیم مجموعه نقاط را به k قسمت تقسیم نماییم و انجام مثلث بندی در هریک از این k ناحیه را برعهده یک پردازنده قرار دهیم، ساخت دیواره مثلث بندی شده بین نواحی نیز بر عهده پردازنده ها و یک پردازنده اصلی که عمل توزیع نقاط را نیز انجام میدهد خواهد بود، هرچند پیاده سازی این الگوریتم، تا حدودی مشکل تر از روش تقسیم و حل خواهد بود ولی دارای بازدهی نسبتا بالایی میباشد، ضمن آنکه مشکل حذف مثلث بندی های غیر دلانی در بلوکها که در الگوریتم ارائه شده در [8] وجود دارد نیز در این الگوریتم وجود نخواهد داشت.

برای درک ساده تر الگوریتم فرض کنید تعداد پردازنده های ما ۹ عدد باشد، ابتدا توسط پردازنده P0 مجموعه نقاط به قسمتهایی با استفاده از خطوط افقی و عمودی تقسیم میگردند و توسط این قسمت بندی به تعداد پردازنده های موجود، مثلا ۸، ناحیه جدید بدست خواهد آمد . هریک از این نواحی به یک پردازنده اختصاص میابد و بجز دو پردازنده اول (سمت چپ ترین) که دیواره آن‌ها برروی پوش محدب نقاط قرار خواهد داشت، ما بقی پردازنده‌ها اقدام به مثلث بندی دیواره خود مینمایند، هر پردازنده در حین مثلث بندی دیواره یالهای نیم صفحه مثبت دیواره را در یک لیست فعال و یالهای نیم صفحه منفی را در لیست فعال دیگری قرار میدهد، و پس از پایان مثلث بندی دیواره یالهای نیم صفحه منفی را به پردازنده با شماره دو واحد کوچکتر از خود ارسال مینماید.

بعبارت دیگر میتوان برای تقسیم بهتر نواحی، توسط یک خط افقی (عمودی) صفحه را به دو نیم تقسیم نمود و مثلث بندی آنرا بر عهده پردازنده صفر قرار داد، نیم صفحه بالایی این دیواره توسط خطوط به $k/2$ ناحیه تقسیم میگردد و نیم صفحه پایین نیز به همین صورت توسط خطوط به $k/2$ ناحیه دیگر تقسیم میگردد (شکل ۲).

پردازنده P0 مثلث بندی برروی دیواره را بصورت افزایشی انجام میدهد و هرگاه یکی از یالهای مثلث ساخته شده دیواره متعلق به یکی از پردازنده ها را قطع نمود، یال قطع کننده به پردازنده مورد نظر فرستاده میشود و مثلث بندی برروی دیواره توسط پردازنده صاحب دیواره آغاز میشود، وقتی پردازنده ای ساخت دیواره مربوط به خود را به پایان رساند یالهای مثلث های دیواره که متعلق به ناحیه خود میباشد (AFL^{+}) را نگهداری کرده و یالهای سمت دیگر دیواره (AFL^{-}) را به پردازنده سمت راست خود ارسال مینماید، برای بالاتر بردن بازدهی این نکته را درالگوریتم درنظر میگیریم که وقتی پردازنده اصلی ساخت دیواره مربوط به یک ناحیه را به پایان رساند آنرا برای پردازنده مربوطه ارسال مینماید، مثلا اگر پردازنده P0 اولین مثلثی که میسازد درناحیه مربوط به پردازنده های ۳ و۴ باشد، چون در هرمرحله که مثلثی اضافه میکند، درمرحله بعد مثلثی درجهت مخالف به آن اضافه میکند، ابتدا دیواره مربوط به نواحی ۳ و۴ توسط این مثلث ساخته خواهد شد و بهمین علت بعد از اتمام این دیواره باید به این دو پردازنده فرستاده شود.
هر پردازنده ای وقتی دیواره مربوطه را از پردازنده P0 و پردازنده سمت چپ خود دریافت نمود مجموعه لیست فعال خود را با استفاده از یالهای دریافت شده میسازد، مجموعه نقاط را به دو نیم مینماید و مثلث بندی را با استفاده از الگوریتم DeWall در این ناحیه انجام میدهد.



شکل ۳. ساخت دیواره توسط پردازنده ها

در ظاهر پیاده سازی ایده ذکر شده در بالا منجر به کارآیی بهتر و زمان اجرای مناسبتر خواهد بود، ولی نکته ای که در پیاده سازی باید در نظر گرفت ساخت دیواره توسط پردازنده های مختلف میباشد، اگر در ابتدا فقط نقاط مربوط به یک ناحیه به پردازنده مربوطه ارسال شوند چون مثلث های مجاور با پوش محدب دارای پهنای زیادی میباشد، ساخت دیواره با مشکل مواجه خواهد شد، به همین علت میتوان در همه مجموعه نقاط یک نیم صفحه را به همه پردازنده های موجود در آن نیم صفحه ارسال نمود و سپس بصورت جداگانه نقاط هر ناحیه را به آن ارسال نمود و یا برای سادگی بیشتر با نشانه‌ای نقطه متعلق به هر ناحیه را مشخص نمود تا ارسال دوباره داده انجام نگردد.
تنها نکته ای که باید به آن توجه نمود یالهای نهایی موجود در لیست فعال در هریک از پردازنده ها بهنگام صدا زدن تابع DeWall خواهد بود، این یالها که از لیست فعال دریافت شده از پردازنده همسایه و پردازنده صفر و لیست فعال ساخته شده در خود پردازنده بوجود می آیند قبل از اجرای تابع DeWall باید کنترل گردند، مشکلی که در این حالت بوجود خواهد آمد اینست که در این لیست فعال یالهای تکراری داشته باشیم، یعنی تعدادی از یالهای لیست فعال فرستاده شده توسط پردازنده صفر در لیست فعال تشکیل شده در خود پردازنده دریافت کننده و لیست فعال

دریافت شده از پردازنده همسایه وجود داشته باشند، بهمین خاطر درصورتیکه یالی در هر دو لیست فعال موجود بود از هر دو لیست حذف میگردد و در لیست نهایی وجود نخواهد داشت.
مشکل دیگر اینست که در یک حالت خاص چون مثلث های ساخته شده روی پوش محدب پهنای زیادی دارند، ممکن است یالی که در لیست فعال پردازنده همسایه خود که قرار است به آن ارسال میکنند قرار میدهند، خارج از ناحیه متعلق به پردازنده همسایه قرار گیرد، و این یال در ناحیه مجاور به همسایه قرار گیرد که برای برطرف نمودن این مشکل نیز هر پردازنده لیست فعال دریافتی از پردازنده همسایه را با محدوده ناحیه خود کنترل کرده و درصورتیکه یالی در ناحیه وجود نداشته باشد غیر مجاز تشخیص داده شده و باید از لیست حذف شود، در شکل ۳ ساخت دیواره توسط پردازنده ها برروی ۱۰۰ نقطه نمایش داده شده است.

۳-۲- پیچیدگی زمانی

الگوریتم DeWall براساس آنچه طراحان الگوریتم ذکر کرده اند در بدترین حالت اجراء بمانند یک الگوریتم افزایشی عمل کرده و در محیط دو بعدی دارای پیچیدگی از مرتبه $O(N^2)$ برای N نقطه درصفحه میباشد هرچند که در عمل این زمان اجرا پایینتر خواهد بود و با استفاده از تکنیک Uniform Grid که در [12] آمده است، عملا زمان اجرا بسیار پایینتر خواهد بود . بطورکلی در محیط d بعدی زمان اجرای الگوریتم از مرتبه پیچیدگی $O(n^{\lceil \frac{d}{2}+1 \rceil})$ خواهد بود.

در الگوریتم ارائه شده در این مقاله اگر تعداد P تا پردازنده داشته باشیم و اگر نقاط بصورت مساوی بین پردازنده ها تقسیم شوند به هر پردازنده به تعداد $\frac{N}{P}$ نقطه خواهد رسید و به این ترتیب برای اجرای الگوریتم DeWall در هریک از این نواحی به زمانی از مرتبه $O(\frac{N^{\lceil \frac{d}{2}+1 \rceil}}{P^{\lceil \frac{d}{2}+1 \rceil}})$ نیاز خواهیم داشت، علاوه براین به زمانی نیز برای ساخت دیواره اصلی نیاز خواهیم داشت که توسط پردازنده اصلی ساخته میشود و در شرایطی که الگوریتم را برای دو بعد اجرا کنیم این زمان از مرتبه پیچیدگی $O(N^{2.5})$ خواهد بود که با توجه به الگوریتم بهینه برای مثلث بندی دلانی که در فضای دو بعدی از مرتبه پیچیدگی $O(N\log N)$ میباشد افزایش سرعت در این الگوریتم در بدترین حالت بصورت زیرخواهد بود :

$$S = \frac{O(N\log N)}{O(\frac{N^2}{P^2}) + O(N^{2.5})} = O(\frac{P^2 \log N}{N^{1.5}})$$

البته در عمل خواهیم دید که زمان اجرا کمتر از زمان ذکر شده خواهد بود، استفاده از Uniform Grid نیز زمان اجرای الگوریتم را بسیار پایینتر می آورد. علاوه براین میتوان برای بهبود زمان اجرا ایده تقسیم نقاط و توزیع نقاط بصورت نابرابر بین پردازنده ها را بکار برد، بصورتیکه نواحی که در منتهی الیهها قرار دارند تعداد نقاط کمتری را در برداشته باشند و یا نواحی سمت راست که نیاز به ساخت دیواره ندارند تعداد نقاط بیشتری را در برگیرند، چرا که مثلث بندی در آنها در مدت زمان کمتری انجام خواهد گرفت، البته برای برای بررسی تعداد نقاط مناسب برای هر ناحیه نیاز به تحلیل دقیقتر و بررسی دقیقتر پیچیدگی زمانی خواهد بود.
نکته دیگر اینکه اگر تعداد پردازنده ها زوج باشد تقسیم نقاط قدری با مشکل مواجه خواهد شد که برای برطرف نمودن آن میتوان یکی از نواحی را کوچکتر درنظر گرفت (مثلا ناحیه ۱) و ناحیه مجاور آنرا بزرگتر درنظر گرفت (مثلا ناحیه ۳) و عمل مثلث بندی در ناحیه ۱ را برعهده پردازنده اصلی قرار داد.
علاوه برمسائلی که ذکر شد شاید اگر هر پردازنده ساخت دو دیواره را برعهده بگیرد (بجز پزدازنده های منتهی الیه) زمان اجرای الگوریتم بطور قابل ملاحظهای پایینتر بیاید، چرا که در اینحالت دیگر پردازنده ای منتظر دریافت لیست

فعال از پردازنده مجاور خود نخواهد بود و همچنین با تقسیم بندی نقاط صفحه بطور مناسبتر نیز میتوان هزینه اجرا را بطور قابل ملاحظه ای پایینتر آورد، چراکه در اینصورت ساخت دیواره توسط پردازنده اصلی در مدت زمان کمتری انجام میپذیرد.

۴- پیاده سازی الگوریتم موازی مثلث بندی دلانی

برای پیاده سازی الگوریتم از توابع کتابخانه ای LEDA استفاده شده است، همچنین پیاده سازی موازی آن نیز با کمک MPI انجام پذیرفته است. بخاطر سادگی بیشتر درپیاده سازی برای ارسال داده ساختارهای موجود در LEDA مانند نقطه، خط و موارد دیگر توسط MPI_Type نوع جدیدی تعریف نشده است، بلکه این داده ها به آرایه هایی از نوع حقیقی با اندازه طول مشخص تبدیل شده و ارسال شده اند. در پیاده سازی انجام شده پردازنده شماره صفر ابتدا اقدام به ناحیه بندی ذکر شده میکند و پس از انجام ناحیه بندی، اطلاعات مربوط به ناحیه متعلق به هر پردازنده را بهمراه مجموعه نقاط به آن ارسال مینماید، ارسال اطلاعات توسط MPI_Send انجام شده است و برای دریافت اطلاعات، بجز در موردی که پردازنده اصلی مثلث بندی های ساخته شده را دریافت مینماید، که در آن از تابع MPI_Gatherv استفاده شده است در بقیه موارد از MPI_Recv استفاده کرده ایم.
برای پیاده سازی نقاط بجای استفاده از داده ساختار نقطه که در LEDA تعریف شده است، داده ساختار جدیدی بنام PointCounter تعریف نموده ایم که علاوه بر آنکه مختصات نقطه را داراست دارای یک شمارنده میباشد که با ساخت یک وجه جدید متقاطع با نقطه یک واحد اضافه گشته و با ساخت یک مثلث جدید برروی یکی از وجوه منتهی به نقطه یک واحد کم میشود و هرگاه این شمارنده به صفر رسید نقطه از فهرست نقاط برای ادامه پردازش حذف میشود، این روش در [1] برای بالاتر بردن بازدهی الگوریتم پیشنهاد شده است. علاوه براین داده ساختار ذکر شده دارای شمارنده دیگری نیزخواهد بود که شماره پردازنده‌ای که نقطه مربوط به ناحیه آن میباشد، در آن ذکر میگردد و این برای جلوگیری از ارسال دوباره نقاط به یک پردازنده و کمتر کردن هزینه تبادل داده انجام شده است.
همچنین برای یالهای لیست فعال داده ساختاری بنام SegmentDirection تعریف شده است که در آن یک یال بهمراه جهتی که قبلا برروی آن مثلث ساخته نشده است ذخیره میشود، که این جهت با دو مقدار ۱- و ۱ نشان داده میشود، اگر ۱- در آن قرار داده شود به این معنی است که مثلثی که قبلا ساخته شده، در زیر یال قرار دارد و مثلثی جدید در بالای یال باید ساخته شود.
در پیاده سازی یک تابع اصلی بنام MakeWall به توابع قبلی افزوده شده که کنترل اصلی اجرای برنامه، ناحیه بندی، ارسال و دریافت یالها و ساخت دیواره برروی خطوط را برعهده خواهد داشت، تعدادی از توابع برنامه توزیع شده قبلی از جمله بافر کردن داده ها و موارد دیگر در این برنامه نیز بکار برده شده اند.
شبیه سازی اجرای موازی الگوریتم برروی یک کامپیوتر نشان دهنده آنست که برای تعداد نقاط زیاد، روش ارائه شده دارای بازدهی بهتری نسبت به موازی سازی ارائه شده بصورت بازگشتی می‌باشد.

۵- نتیجه گیری

در این مقاله نوعی موازی سازی برای الگوریتم Dewall ارائه گردید، این الگوریتم در سال ۱۹۹۸ مطرح گردیده است و برمبنای روش تقسیم و حل میباشد، ویژگی بارز این الگوریتم علاوه بر بازدهی مناسب در اجرا، بسط راحت آن به محیط های با ابعاد بالاتر میباشد. موازی سازی ارائه شده نسبت به روش اختصاص پردازنده بطور پویا دارای بازدهی مناسبتری خواهد بود، ضمن آنکه در برابر الگوریتم ارائه شده در [8] علاوه برآنکه بازدهی قابل قبولی دارد، قابلیت بسط به محیط با ابعاد بیشتر را خواهد داشت و همچنین مشکل حذف مثلث های غیر دلانی و چهار نقطه برروی یک دایره را که در آن وجود دارد نخواهد داشت.

ضمن آنکه این الگوریتم خصوصیت ساخت مثلثهای زیربلوکها بطور جداگانه را حفظ مینماید و در آن هر پردازنده تنها با ۲ یا ۳ پردازنده دیگر تبادل اطلاعات خواهد داشت، همچنین با استفاده از تکنیکهایی که در بالا مطرح شد میتوان زمان اجرای الگوریتم را درحالت عملی تا حد بسیاری پایین ترآورد.

مراجع

[1]P.Cignoni,C.Montani,R.Scopigno,"DeWall:A Fast Divide & Conquer Delaunay Triangulation Algorithm in E^d",computer Aided design ,1998.

[2]P.Su,R.Scot Drysdale,"A Comparision of Sequential Delaunay Triangulation Algorithms", 11th ACM Computational Geometry Conference proceeding,1995.

[3] P.Su,"Efficient Parallel Algorithms for Closest Point Problems".Technical Report CS—1994—22.

[4].P.Cignoni,C.Montani,R.Perego,R.Scopigno,"Parallel 3D Delaunay Triangulation"Computer Graphics Forum , 1993.

[5] D.Ashton,W.Gropp,Installation and user's Guide to MPICH, www -unix.mcs.anl.gov/mpi/.

[6] MPI Primer/Developing with LAM, www.csd.uch.gr/~hy443/MPI/lam61.pdf.

[7] Ian Foster,Designing and Building parallel programs, Addison Wesley; 1st edition , 1995.

[8] M .B .Chen , T.R .Chuang , J. J. Wu , Parallel 2D Delaunay Triangulation in HPF and MPI , Proceeding in 15th international Parallel and distributed processing , 2001

[9] LEDA User Manual , www.algorithmic-solutions.com

[10] G.Hristescu,"Parallel Triangulation of a set of points for Coarse Grained Multicomputers".

[11] P.Magillo,E.Puppo,"Algorithms for parallel Terrain Modelling and Visulization".

[12] T.P.Fang and L.A. Piegl ,Delaunay triangulationusing a uniform Grid , IEEE Computer Graphics and Applications , 1993.

[13] M.BChen, T.R Chaung, J.J.Wu Efficient Parallel implementation, of near Delaunay triangulation with high performance fortran, Concurrency & computation: Practice and Experience, p.1143-1159 2004.