بسم الله الرحمن الرحیم

دانشکده : آموزش های الکترونیکی
گروه : مهندسی کامپیوتر

افزایش سرعت الگوریتم گرده افشانی گل بواسطه پردازش کارت گرافیک

دانشجو : مرتضی عزیزی ثالث

استاد راهنما :
(دکتر حمید حسن پور)

پایان نامه ارشد جهت اخذ درجه کارشناسی ارشد

ماه و سال انتشار :
اسفند ماه ۱۳۹۴

دانشگاه صنعتی شاهرود

دانشکده : مرکز آموزش های الکترونیکی

گروه : مهندسی کامپیوتر

پایان نامه کارشناسی ارشد آقای مرتضی عزیزی ثالث

تحت عنوان: افزایش سرعت الگوریتم گرده افشانی گل بواسطه پردازش کارت گرافیک

در تاریخ ..... توسط کمیته تخصصی زیر جهت اخذ مدرک کارشناسی ارشد مورد ارزیابی و با درجه ..... مورد پذیرش قرار گرفت.

اساتید راهنما	امضاء	اساتید مشاور	امضاء
نام و نام خانوادگی :		نام و نام خانوادگی :	
نام و نام خانوادگی :		نام و نام خانوادگی :	

اساتید داور	امضاء	نماینده تحصیلات تکمیلی	امضاء
نام و نام خانوادگی :		نام و نام خانوادگی :	
نام و نام خانوادگی :			
نام و نام خانوادگی :			
نام و نام خانوادگی :			

تقدیم به

به مادرم و روح پدرم که برای من زحمات زیادی کشیدند.

سپاسگزاری :

با سپاس از جناب آقای دکتر حمید حسن پور به عنوان استاد راهنما که همواره اینجانب را مورد لطف و محبت خود قرار داده اند.

تعهد نامه

اینجانب مرتضی عزیزی ثالث دانشجوی دوره کارشناسی ارشد رشته کامپیوتر گرایش هوش مصنوعی دانشکده آموزش های الکترونیکی دانشگاه صنعتی شاهرود نویسنده پایان نامه افزایش سرعت الگوریتم گرده افشانی گل بواسطه پردازش کارت گرافیک، تحت راهنمائی دکتر حمید حسن پور متعهد می شوم .

- تحقیقات در این پایان نامه توسط اینجانب انجام شده است و از صحت و اصالت برخوردار است
- در استفاده از نتایج پژوهشهای محققان دیگر به مرجع مورد استفاده استناد شده است .
- مطالب مندرج در پایان نامه تاکنون توسط خود یا فرد دیگری برای دریافت هیچ نوع مدرک یا امتیازی در هیچ جا ارائه نشده است .
- کلیه حقوق معنوی این اثر متعلق به دانشگاه صنعتی شاهرود می باشد و مقالات مستخرج با نام « دانشگاه شاهرود » و یا « Shahrood University of Technology » به چاپ خواهد رسید .
- حقوق معنوی تمام افرادی که در به دست آمدن نتایج اصلی پایان نامه تأثیرگذار بوده اند در مقالات مستخرج از پایان نامه رعایت می گردد.
- در کلیه مراحل انجام این پایان نامه ، در مواردی که از موجود زنده (یا بافتهای آنها) استفاده شده است ضوابط و اصول اخلاقی رعایت شده است .
- در کلیه مراحل انجام این پایان نامه، در مواردی که به حوزه اطلاعات شخصی افراد دسترسی یافته یا استفاده شده است اصل رازداری ، ضوابط و اصول اخلاق انسانی رعایت شده است .

مرتضی عزیزی ثالث

۱۳۹۴

مالکیت نتایج و حق نشر

- **کلیه حقوق معنوی این اثر و محصولات آن (مقالات مستخرج ، کتاب ، برنامه های رایانه ای ، نرم افزار ها و تجهیزات ساخته شده است) متعلق به دانشگاه صنعتی شاهرود می باشد . این مطلب باید به نحو مقتضی در تولیدات علمی مربوطه ذکر شود .**

چکیده

الگوریتم‌های تکاملی(Evolutionary algorithms)، الگوریتم هایی الهام گرفته از طبیعت هستند که جهت رسیدن به جواب بهینه، عمل جستجو را از چندین نقطه در فضای جواب آغاز می کنند و اغلب در مسائلی استفاده می شود که به دلیل پیچیدگی و پارامترهای بسیار مسئله، راه حل های عادی و متعارف برای آنها چاره ساز نیستند.

الگوریتم‌های تکاملی با الهام از طبیعت و قوانین حاکم بر آن سعی می‌کنند راه‌حل بهینه مسائل سخت و دشوار را با دقت مناسب محاسبه نمایند. تاکنون الگوریتم‌های تکاملی مختلفی بر اساس رفتار اجتماعی جانداران، رفتار زیستی موجودات و پدیده‌های فیزیکی ارایه شده است. گرده-افشانی گلها یک فرآیند جالب است که تعداد زیادی از گونه‌های گیاهان برای تولید مثل و بقای خود از آن استفاده می‌نمایند و پژوهشگران از این رفتار گیاهان برای ایجاد یک الگوریتم تکاملی تحت نام گرده‌افشانی گلها استفاده نمودند.
یافتن جوابهای بهینه یک مسئله بهینه‌سازی با افزایش ابعاد مسئله به کمک الگوریتم‌های تکاملی از جمله الگوریتم گرده‌افشانی گل چالش برانگیز و دشوار می‌باشد و این احتمال وجود دارد الگوریتم گرده‌افشانی گل به جای بهینه سراسری به بهینه‌های محلی مسئله همگرا شود که برای رفع این مشکل نیاز است اندازه جمعیت اولیه و تکرار الگوریتم گرده‌افشانی گل افزایش یاید و این افزایش، زمان اجرای الگوریتم را به شدت افزایش می‌دهد.
در این پژوهش تلاش شده است تا هر کدام از اعضای جمعیت اولیه را به شکل موازی و در واحد پردازنده کارت گرافیک که دارای هسته‌های پردازشی به مراتب بیشتری نسبت به پردازنده اصلی می‌باشد، استفاده نمایم. نتایج آزمایشات و شبیه‌سازی‌ها نشان می‌دهد نسخه موازی الگوریتم گرده‌افشانی گل به کمک فناوری موازی سازی کارت گرافیک(کودا) سرعت اجرای بیشتری نسبت به الگوریتم استاندارد گرده‌افشانی گل دارد.

واژگان کلیدی:
الگوریتم‌های تکاملی، الگوریتم گرده‌افشانی گل، واحد پردازنده کارت گرافیک، **کودا**، نسخه موازی الگوریتم گرده‌افشانی گل

فهرست مطالب

عنوان..... صفحه

فهرست اشکال

عنوان..... صفحه

فهرست جداول

عنوان..... صفحه

فهرست روابط

فصل ۱- مقدمه و بیان مسئله

۱-۱- مقدمه

پدیده‌های طبیعی بخصوص جانداران و موجودات زنده که دارای فرآیندهای زیستی می‌باشند در طی زمان و با گذشت میلیون‌ها سال توانسته‌اند با شرایط و محیط اقلیمی خود به سازگاری برسند و به حیات ادامه دهند. تکامل[۱] یک پدیده یک شبه و گذار نیست بلکه یک پدیده زمان‌بر است که باعث می‌شود به مرور زمان گونه‌های مختلفی که شایستگی بیشتری در بقاء دارند به ادامه نسل بپردازند. در اصول تکاملی داروین[۲] و اصل انتخاب طبیعی[۳] به این موضوع تاکید می‌شود که افراد شایسته[۴] یک جمعیت، شانس بقای بیشتری از افراد غیر شایسته[۵] آن جمعیت دارند. گونه‌های مختلف زیستی با استفاده از اصل انتخاب طبیعی توانسته‌اند خود را با محیط پیرامون تطبیق داده و شرایط ادامه نسل را برای خود محیاء سازند. بشر با آموختن از طبیعت و اصول حاکم بر آن توانسته است بر بسیاری از چالش‌های پیرامون غلبه نماید و با بکارگیری این اصول در زندگی، شانس بقای خود را افزایش دهد[۱]. مسائل مختلفی پیرامون ما وجود دارد که دارای راه‌حل واحدی نبوده بلکه راه‌حلهای مختلفی دارد که ارزش و میزان بهینه بودن آنها با هم متفاوت است و تلاش برای یافتن راه‌حل بهینه[۶] یکی از شاخه‌های هوش محاسباتی را به نام روش‌های جستجو[۷] ایجاد نموده است. الگوریتم‌های تکاملی[۸] یکی از روش‌های جستجوی شبه‌تصادفی و البته هوشمندانه به کمک توابع اکتشافی[۹] می‌باشد که از اصول طبیعت و تکاملی الهام گرفته شده است به عبارتی بهتر الگوریتم‌های تکاملی دسته‌ای از الگوریتم‌های الهام گرفته از طبیعت و قوانین ناظر بر آن و جهت مسائل سخت و دشوار[۱۰] بهینه‌سازی[۱۱] می‌باشند[۲]. باشند[۲]. الگوریتم‌های تکاملی جهت حل مسائل سخت و دشوار امروزی که قالب روش‌های سابق مانند گرادیان[۱۲] و سیپملکس[۱۳] به خوبی آنها را حل نمی‌کنند از تکنیک‌های مختلفی نظیر الهام از رفتار حشرات، جانوران، موجودات تک سلولی، سازوکارهای تکاملی سلولی، رفتار پدیده‌های فیزیکی، رفتار اجتماعی و سیاسی، رفتارهای روزمره انسان‌ها و موارد دیگر الهام گرفته شده است[۳].

الهام و تقلید از رفتار گیاهان برای خلق الگوریتم‌های تکاملی یک ایده نو و جدید است که در آن رفتارهای زیستی گیاهان دست مایه ابداع الگوریتم‌های جستجو از نوع تکاملی قرار داده می‌شود. از جمله الگوریتم‌های تکاملی مبتنی بر رفتار گیاهان می‌توان الگوریتم فتوسنتز[۱۴][۴]، الگوریتم علف‌های

---

[1] Evolution
[2] Darwin
[3] natural selection
[4] Fitness
[5] Non-Fitness
[6] Optimal solution
[7] Search techniques
[8] Evolutionary algorithms
[9] Heuristic
[10] non-deterministic polynomial-time hard
[11] Optimization problems
[12] Gradient
[13] Simplex
[14] Photosynthetic algorithm (PA)

هرز[1][۵] و الگوریتم گرده‌افشانی گل‌ها[2][۶] را برشمرد. الگوریتم‌های تکاملی از هر دسته و هر نوع تقلیدی که از آن الهام گرفته شده باشند دارای چند ویژگی مشترک می‌باشند که از جمله آنها می-توان به افزایش دقت یافتن جوابها با افزایش تعداد تکرار و جمعیت اولیه، احتمال گرفتار شدن در بیشینه محلی[3] و کندی یافتن جوابهای بهینه و نیمه‌بهینه در حالتی که تعداد تکرار و جمعیت اولیه[4] زیاد است، اشاره نمود. در این پژوهش جهت افزایش سرعت و شتاب[5] الگوریتم تکاملی گرده‌افشانی گل‌ها با استفاده از معماری موازی کودا[6][۷] یک روش پیشنهادی ارایه خواهد شد تا یک نسخه سریع-تر و موازی از این الگوریتم بر پایه پردازنده کارت گرافیک[7] توسعه داده شود. در این فصل مقدمات، تعاریف، بیان مسئله، فرضیات، سوالات، چالش‌ها و محدودیت‌های این پژوهش به همراه تعداد دیگر از مطالب پیش‌نیاز این پایان نامه ارایه خواهد شد.

[1] Photosynthetic algorithm (PA)
[2] Flower pollination algorithm (FPA)
[3] Local optimal
[4] initial population
[5] Speedup
[6] Compute Unified Device Architecture (CUDA)
[7] Graphics Processing Unit (GPU)

۱-۲- تعاریف اولیه

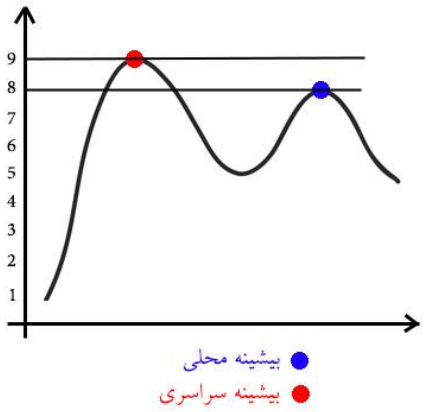
در این فصل به تعدادی از تعاریف بکار رفته در این پایان‌نامه به شکل مختصر پرداخته می‌شود تا خواننده در بخش‌های بعدی با دید خوبی قادر به فهم و دنبال کردن مطالب پایان‌نامه باشد.

۱-۲-۱- الگوریتم تکاملی

الگوریتم تکاملی به دسته‌ای از الگوریتم‌های جستجوی هوشمند و الهام گرفته از طبیعت گفته می‌شود که راه‌حلهای مسئله با پدیده‌های طبیعی نظیر کروموزوم، ذرات، گل و غیره مدلسازی می‌شود و با تکرار گام‌های الگوریتم در نهایت بهترین عضو جمعیت به عنوان جواب بهینه مسئله در نظر گرفته می‌شود[۱].

۱-۲-۲- بیشینه و کمینه تابع

جهت ارزیابی و سنجش شایستگی جمعیت یک الگوریتم تکاملی نیاز است که هر کدام از اعضای جمعیت توسط یک تابع هدف[1] مورد ارزیابی قرار گیرند. بسته به ماهیت مسئله بهینه‌سازی که به شکل تابع هدف مدلسازی می‌شود، مسئله مورد نظر از نوع بیشینه‌یابی یا کمینه‌یابی است. پژوهشگران علاقمند می‌باشند که بهینه‌های سراسری یک مسئله را پیدا نمایند هر چند این امکان وجود دارد که روش‌های تکاملی در بهینه‌های محلی گرفتار شده و آنها را به عنوان جواب بهینه عرضه نمایند. در شکل(۱-۱) مفهوم بیشینه محلی و سراسری یک تابع هدف به تصویر کشیده شده است[۸].



شکل ۱-۱: تابع هدف با چهار بیشینه سراسری و یک کمینه سراسری

[1] Objective Function

بیشینه سراسری[1] به نقطه‌ای در فضای تابع هدف گفته می‌شود که سایر نقاط تابع هدف دارای مقادیری کمتر از این نقطه باشند و در تعریفی مشابه، کمینه سراسری[2] به نقطه‌ای در فضای تابع هدف گفته می‌شود که سایر نقاط تابع هدف دارای مقادیری بیشتر از این نقطه باشند.

۱-۲-۳- پردازنده کارت گرافیک

کارتهای گرافیک امروزی برای انجام پردازش‌های گرافیکی و اخیراً محاسباتی دارای یک پردازنده مستقل و مجزاء تحت نام پردازنده کارت گرافیک[3] می‌باشند که بر خلاف پردازنده اصلی سیستم[4] یک پردازنده اختصاصی و در اختیار کارت گرافیک است که معماری آن متفاوت از معماری پردازنده اصلی است و دارای هزاران هسته[5] پردازشی است که یک معماری موازی و قوی بر انجام پردازش‌های موازی فراهم می‌آورد. در برنامه نویسی کودا دو اصطلاح رایج سخت افزاری میزبان[6] و دستگاه[7] از مفاهیم کلیدی محسوب می‌شوند. در شکل(۱-۲) مفهوم میزبان در برنامه‌نویسی موازی کودا نشان داده شده است. به طور کلی میزان همان پردازنده اصلی و حافظه متصل به آن می‌باشد[۹].

شکل ۱-۲: سخت افزار میزبان در برنامه نویسی کودا

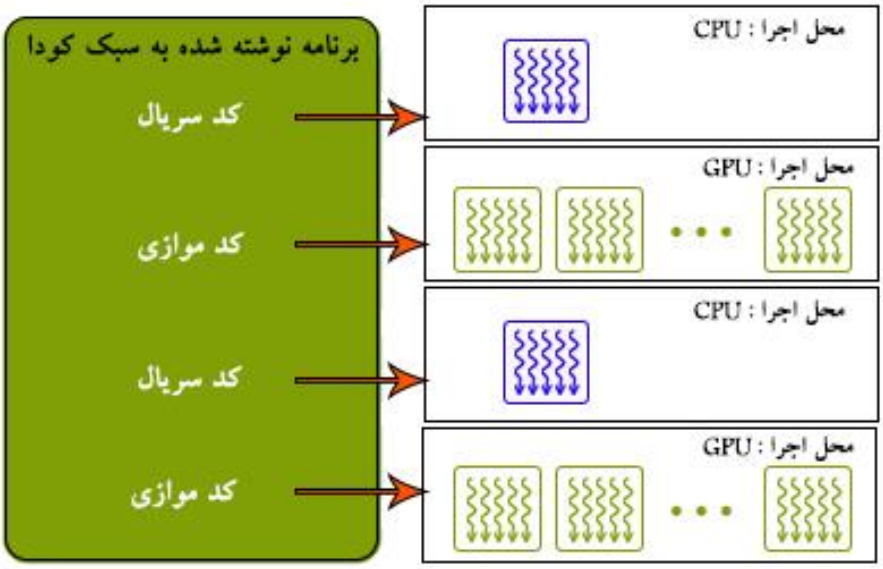
دستگاه در معماری کودا همان مفهوم کارت گرافیک است که شامل پردازنده گرافیکی، اتصالات، حافظه کارت گرافیک و غیره است. در شکل(۱-۳) یک نما از سخت‌افزار دستگاه در برنامه‌نویسی موازی کودا نشان داده شده است[۹].

[1] Global Maximum
[2] Global Minimum
[3] GPU
[4] CPU
[5] Core
[6] Host
[7] Device

شکل ۱-۳: سخت افزار دستگاه در برنامه نویسی کودا

۱-۲-۴- کودا

کودا یک چارچوب نرم‌افزاری است که می‌توان آن را به عنوان یک کتابخانه و لایه بر روی سخت افزار پردازنده گرافیکی در نظر گرفت که هدف آن سهولت دسترسی برنامه‌نویسان به ساختارهای موازی در این پردازنده اختصاصی برای انجام محاسبات عموماً غیرگرافیکی به شکل موازی می‌باشد. در شکل(۱-۴) نحوه اجرای یک برنامه موازی در این معماری به تصویر کشیده شده است[۱۰].



شکل ۱-۴: روند اجرای یک کد در برنامه نویسی به سبک کودا

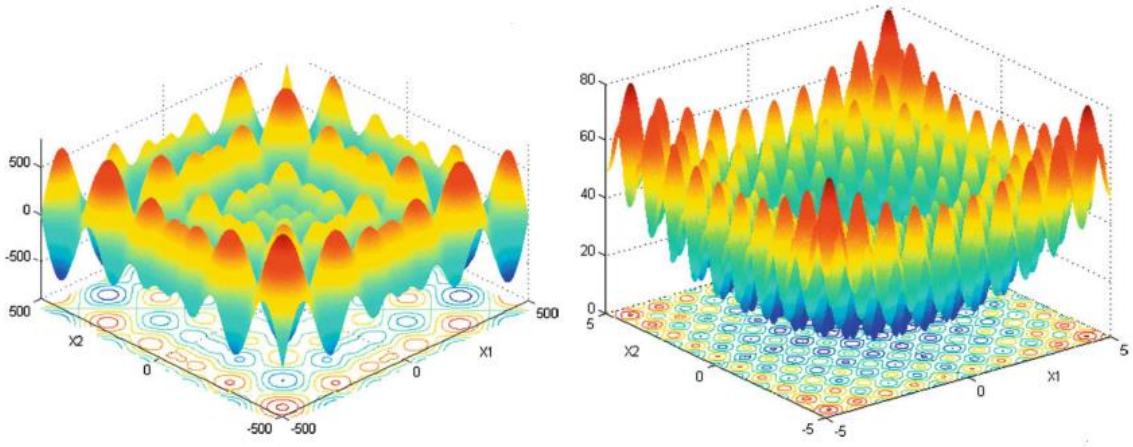
در این سبک برنامه‌نویسی کدهای برنامه می‌تواند بین پردازنده اصلی و گرافیکی سویچ نموده و زمانی که برنامه در پردازنده اصلی در حال اجراء است توسط تعداد نخ‌های[1] محدودی به اجراء می‌شود و این در حالی است که همین برنامه در پردازنده گرافیکی با تعداد بسیار بیشتری نخ اجراء می‌شود. همانگونه که در این شکل نشان داده است یک برنامه با تعداد نخ‌های بیشتری در پردازنده گرافیکی

[1] Threads

نسبت به پردازنده اصلی اجراء می‌شود و به عبارتی بهتر سطح توازی‌سازی پردازنده گرافیکی از پردازنده اصلی بیشتر است و آن به دلیل معماری موازی و تعداد هسته‌های محاسباتی پردازنده گرافیکی است.

۱-۲-۵- تابع ارزیابی

یکی از روش‌های تست دقت و همگرایی الگوریتم‌های تکاملی استفاده از توابع ارزیابی ریاضی[1] به عنوان معیارهای ارزیابی استاندارد می‌باشند. در بسیاری از این توابع بیشینه و کمینه فراوانی وجود دارد که به طور عام الگوریتم تکاملی سعی در یافتن کمینه‌های سراسری این توابع دارد. الگوریتم تکاملی که با تکرار کمتری بتوانند با دقت بیشتری به کمینه‌های سراسری این توابع نزدیک شوند به اصطلاح دقیق و همگراتر نامیده می‌شوند. در شکل(۱-۵) دو نمونه از توابع ارزیابی ریاضی در فضای سه بعدی نشان داده شده است که در نقطه (0,0) دارای کمینه سراسری به مقدار صفر می-باشند[۱۱].

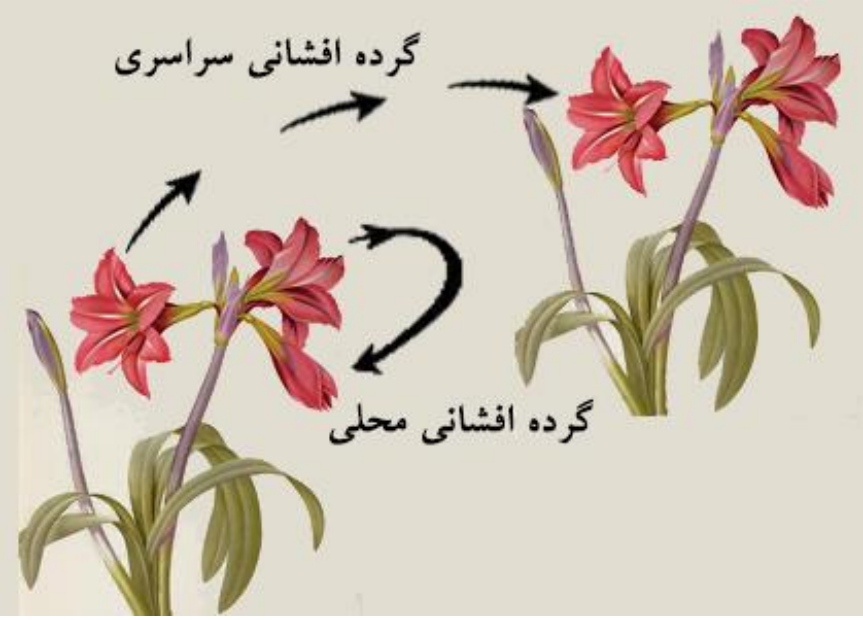


شکل ۱-۵: دو نمونه از توابع ارزیابی ریاضی [۱۱]

۱-۲-۶- گرده افشانی در گل

یک گل جهت تکثیر نیاز به تولید دانه و میوه دارد که بدون گرده‌افشانی گلهای آن ممکن نیست. در گیاهان دو نوع گرده‌افشانی دگرالقاحی[2] و خودالقایی[3] دیده می‌شود. در نوع دگرالقایی گرده‌های یک گل از یک گیاه بر روی یک گل از گیاه دیگر قرار می‌گیرد و در نوع خودالقایی گرده‌های گل یک گیاه بر روی یکی از گل‌های آن گیاه قرار می‌گیرد. در شکل(۱-۶) گرده افشانی سراسری و محلی نشان داه شده است [۱۲].

[1] Benchmark function
[2] Cross Pollination
[3] Self Pollination



شکل ۱-۶: گرده افشانی سراسری و محلی

همانگونه که در شکل مشخص شده است گرده‌افشانی دگرالقایی یا سراسری بین دو گیاه مختلف صورت گرفته و بیشتر برای حالتی است که گرده‌ها بخواهند مسافت بیشتری را به کمک حشرات یا باد طی نمایند و در نوع خودالقایی حرکت گرده‌ها به کمک حشرات، باد یا تکان‌های گیاه انجام می-گیرد.

۱-۳- بیان مسئله

اکثر مسائل موجود و صنعت به گونه‌ای است که پژوهشگران علاقمند می‌باشند که بهترین طرحها را با حداقل هزینه‌ها[۱] پیاده‌سازی نمایند. به عبارت دیگر بسیاری از مسائل پیرامون ما در قالب مسائل بهینه‌سازی[۲] مدلسازی می‌شوند که دارای راه‌حلهای مختلفی از نظر میزان بهینه بودن[۳] می‌باشند. یک مسئله بهینه‌سازی عموماً به شکل روابط و معادلات ریاضی و در قالب تابع هدف[۴] ارایه می‌شود که هدف نهایی یافتن بیشینه یا کمینه این تابع بسته به ماهیت مسئله می‌باشد. مسائل بهینه‌سازی واقعی به طور معمول پیچیده و غیرخطی[۵] می‌باشند که حل آنها را توسط روش‌های ریاضی مانند گرادیان[۶] و سیمپلکس[۷] سخت و تا حدی غیر ممکن می‌نماید[۱] . روش‌های تکاملی نوع جدیدی از محاسبات تکاملی جهت حل مسائل سخت و دشوار بهینه‌سازی محسوب می‌شوند که روز به روز به تعداد آنها

[1] Cost
[2] Optimization problems
[3] Optimal
[4] Objective function
[5] Nonlinear
[6] Gradient
[7] Simplex

افزوده می‌شود. در الگوریتم‌های تکاملی[1] یک پدیده طبیعی و تکاملی که بیشتر ماهیت زیستی دارد دست مایه تقلید قرار گرفته تا به کمک این مکانیزم موجود در طبیعت و شیوه تکاملی آن، یک مسئله دشوار به سهولت حل شود. تاکنون الگوریتم‌های تکاملی فراوانی بر پایه رفتار جانداران، گیاهان، حشرات، پدیده‌های فیزیکی و غیره مدلسازی و ارایه شده است[۳]. به عنوان نمونه می‌توان به رفتار اجتماعی کرم‌های شب‌تاب[2] در یافتن غذا و جفت به کمک سیگنال‌های نوری آنها اشاره نمود که یک نمونه از هوش دسته جمعی[3] حشرات محسوب می‌شود[۱۳]. از نمونه‌های دیگر الگوریتم‌های تکاملی می‌توان به رفتارهای جالب گیاهان در ادامه نسل و زاد و ولد اشاره نمود. در این گونه الگوریتم‌های مبتنی بر رفتار گیاهان نظیر الگوریتم گرده‌افشانی گل[4] و علف‌های هرز[5] غلبه گیاهان شایسته‌تر بر گیاهان ضعیف برای ادامه نسل الهام بخش شیوه جسجو و حل مسئله بوده است[۶و۵]. الگوریتم‌های تکاملی فقط بر پایه رفتار موجودات زنده و جاندار نمی‌باشد بلکه پدیده‌های فیزیکی مانند الگوریتم قطرات آب[6][۱۴] و چرخه آب[7][۱۵] در طبیعت نیز الهام بخش شیوه‌های حل مسئله بوده است.

الگوریتم‌های تکاملی علیرغم تفاوت در مراحل و شیوه تقلید از پدیده‌های طبیعی دارای اشتراکی نظیر تولید جمعیت اولیه[8] به عنوان راه‌حل‌های ابتدایی مسئله، تعداد گام و تکرار، به روز رسانی جمعیت، انتخاب افراد شایسته[9] و غیره می‌باشند[۱۶]. الگوریتم‌های تکاملی جهت همگراء شدن و رسیدن به جوابهای بهینه سراسری[10] نیاز دارند که فضای جستجوی مسئله را به خوبی مورد جستجو قرار دهند. افزایش تعداد جمعیت اولیه می‌تواند گرفتار شدن در بهینه‌های محلی[11] را کاهش دهد و از طرفی افزایش تعداد تکرار الگوریتم تکاملی می‌تواند دقت جوابهای مسئله را افزایش دهد. به طور کلی افزایش تکرار و اعضای جمعیت اولیه در یک الگوریتم تکاملی باعث می‌شود افراد شایسته جمعیت به جوابهای بهینه سراسری با دقت بیشتری همگراء شوند. متاسفانه افزایش تعداد اعضای جمعیت اولیه و تعداد تکرار الگوریتم تکاملی باعث می‌شود که تعداد محاسبات این الگوریتم‌ها افزایش یابد و بر روی زمان اجرای آن الگوریتم تاثیر منفی بگذارد. الگوریتم گرده‌افشانی گل‌ها به عنوان یک الگوریتم زیستی که بر اساس رفتار گرده‌افشانی و تولید مثل گیاهان پایه ریزی شده است از این قاعده مستثنی نیست. یکی از خصوصیات الگوریتم گرده‌افشانی گل‌ها یا هر الگوریتم تکاملی توانایی ذاتی هر کدام از اعضای جمعیت اولیه برای جستجوی مستقل فضای مسئله[12] یا تابع هدف می‌باشد. یکی از شیوه‌های افزایش سرعت الگوریتم‌های تکاملی، اجرای آنها به شکل موازی است[۱۹و۱۸و۱۷]. در این شیوه هر کدام از

[1] Evolutionary algorithms
[2] Firefly
[3] Swarm intelligence
[4] flower pollination algorithm
[5] invasive weed optimization (IWO) algorithm
[6] Intelligent Water Drops algorithm
[7] water cycle algorithm (WCA)
[8] initial population
[9] Filtness
[10] Global optimal
[11] Local optimal
[12] Problem Space

اعضای جمعیت به شکل موازی فضای مسئله را مورد جستجو قرار داده که می‌تواند زمان اجراء و رسیدن به بهینه سراسری را کاهش دهد. تاکنون شیوه‌های مختلفی جهت موازی سازی الگوریتم‌ها ارایه شده است که برنامه‌نویسی تحت پردازنده گرافیکی[1] و با چارچوب کودا[2] یکی از شیوه‌های نوین و جدید موازی سازی محسوب می‌شود. در پردازنده‌های گرافیکی هسته‌های محاسباتی بسیار بیشتری نسبت به پردازنده اصلی[3] وجود دارد که قادرند عملیات‌های ساده را به تعداد بسیار زیاد انجام دهند. در این پژوهش تلاش می‌شود که هر گل به عنوان عضو جمعیت الگوریتم گرده‌افشانی گل‌ها به شکل مستقل و موازی درون یک هسته[4] پردازشی پردازنده گرافیکی به اجراء گذاشته شود تا روند کلی الگوریتم گرده‌افشانی گل‌ها به شکل موازی و با سرعت بیشتر صورت پذیرد. برای ارزیابی سرعت و شتاب[5] الگوریتم پیشنهادی از توابع ارزیابی ریاضی[6] استفاده نموده و نسخه پیشنهادی در نهایت با نسخه سریال الگوریتم گرده‌افشانی گل‌ها مورد مقایسه قرار خواهد گرفت.

۱-۴- انگیزه و اهداف پژوهش

انگیزه و اهداف این پژوهش را می‌توان در موارد ذیل خلاصه نمود:

- بهبود سرعت الگوریتم گرده‌افشانی گل‌ها به کمک معماری موازی کودا
- حل دقیق‌تر مسائل بهینه‌سازی با افزایش جمعیت اولیه و تعداد تکرار بیشتر بدون افزایش چشمگیر زمان اجرای الگوریتم گرده‌افشانی گل‌ها

۱-۵- فرضیات تحقیق

- سرعت روش پیشنهادی از الگوریتم استاندارد گرده‌افشانی گل‌ها بیشتر است.
- زمان اجرای الگوریتم پیشنهادی و الگوریتم استاندارد گرده‌افشانی گل‌ها با افزایش جمعیت اولیه و تکرار الگوریتم افزایش می‌یابد ولی این افزایش در حالت غیر موازی به مراتب بیشتر از نسخه موازی است.
- شتاب و سرعت الگوریتم موازی گرده‌افشانی گل‌ها با افزایش تعداد گل‌ها و تعداد تکرار افزایش می‌یابد.

[1] Graphics Processing Unit (GPU)
[2] Compute Unified Device Architecture(CUDA)
[3] CPU
[4] Core
[5] Speedup
[6] Benchmark function

۱-۶- داده های ارزیابی

داده‌های که در این پژوهش بکار گرفته می‌شوند مجموعه‌ای از نقاط هندسی در فضاهای چندبعدی است که با گل کدینگ می‌شوند و تعدادی از توابع ارزیابی ریاضی مشابه منبع[۲۰] را جهت ارزیابی و مقایسه روش پیشنهادی با نسخه استاندارد الگوریتم گرده‌افشانی گل‌ها استفاده خواهد شد.

۱-۷- محدودیت های تحقیق

مسائل بهینه‌سازی واقعی عموماً پیچیده و غیرخطی می‌باشند که تابع هدف آنها دارای شکل پیچیده و با بهینه‌های محلی و احیاناً سراسری متعدد می‌باشند. هر الگوریتم تکاملی به دلیل ماهیت و ذات خود می‌تواند در بهینه محلی گرفتار شود که الگوریتم پیشنهادی ما نیز به علت اینکه از خانواده الگوریتم-های تکاملی موازی است از این قاعده مستثنی نیست. از طرفی اجرای روش پیشنهادی و الگوریتم‌های موازی کودا نیازمند وجود سخت‌افزار گرافیکی مناسب و منطبق با شرکت انویدیا می‌باشد که اجرای الگوریتم پیشنهادی را بر روی سیستم‌های مختلف محدود می‌سازد و از طرفی برنامه‌نویسی کودا یک سبک برنامه‌نویسی جدید است که نیاز است برنامه‌نویس با معماری سخت‌افزار کارت گرافیک تا حدود زیادی آشنا باشد که این مسئله کار را دشوار می‌نماید.

۱-۸- کاربردهای تحقیق

این پژوهش در راستای مسائل مرتبط با بهینه‌سازی و هر مسئله‌ای که بتوان آن را در قالب تابع هدفی از نوع بیشینه و کمینه تعریف نمود کاربرد دارد. از طرفی چون اکثر مسائل واقعی در صنعت به نوعی یک مسئله بهینه‌سازی محسوب می‌شوند می‌توان از این پژوهش برای حل سریع‌تر مسائل بهینه‌سازی که با روش‌های سنتی مانند گرادیان به سختی حل می‌شوند استفاده نمود.

۱-۹- متغیر و معیارهای ارزیابی پژوهش

در این پژوهش تعدادی از متغیرها وابسته و مستقل می‌باشند به عنوان نمونه مهمترین متغیرهای این تحقیق که مستقل محسوب می‌شوند عبارتند از :

- تعداد تکرار
- تعداد گل‌ها
- متغیرها و پارامترهای الگوریتم گرده‌افشانی گل
- تعداد اعضای جمعیت الگوریتم گرده‌افشانی گل
- و غیره

مهمترین متغیرهای این تحقیق که وابسته محسوب می‌شوند عبارتند از :

- شتاب و سرعت که به تکرار و تعداد اعضای جمعیت اولیه و مواردی دیگر مانند نوع تابع ارزیابی و غیره بستگی دارد.
- میزان خطا که به تکرار و تعداد جمعیت اولیه و غیره بستگی دارد.
- و غیره

مهمترین معیار ارزیابی این تحقیق عبارتند از :

- زمان اجراء در پردازنده اصلی
- زمان اجراء در پردازنده گرافیکی
- شتاب الگوریتم که حاصل تقسیم زمان اجراء در پردازنده اصلی به پردازنده گرافیکی است.

۱-۱۰- روش تحقیق و پژوهش

در مرحله اول به جمع آوری اطلاعات مورد نیاز از کتاب ها و مقالات مرتبط با برنامه نویسی موازی ، پردازنده گرافیکی، چارچوب کودا، الگوریتم‌های تکاملی، الگوریتم گرده‌افشانی پرداخته می‌شود و در ادامه سیستم پیشنهادی با استفاده از کامپایلر ویژال استودیو و زبان سی‌شارپ پیاده‌سازی می‌شود و در نهایت روش پیشنهادی را از نظر زمان و دقت با تعدادی از روش‌های تکاملی مورد مقایسه قرار خواهد گرفت. نوع مطالعات این پژوهش بیشتر بر پایه مطالعات کتابخانه‌ای انجام شده است.

۱-۱۱- ساختار پژوهش

این پژوهش در ۶ فصل گرده‌آوری شده است که در فصل اول به مقدمه، بیان مسئله، تعاریف کلیدی، فرضیات و پرسش‌های تحقیق و موارد پرداخته شده است. در فصل دوم ادبیات موضوعی این پژوهش مواردی نظیر آشنایی با الگوریتم گرده‌افشانی گل، معماری موازی پردازنده گرافیکی و مفاهیم اولیه کودا بررسی شده است تا خوانند با این معماری آشنا شود. در فصل سوم مروری بر چندین پژوهش مرتط با الگوریتم‌های تکاملی و موازی‌سازی آنها شده است. در فصل چهارم روش پیشنهادی مورد بررسی قرار گرفته شده است. در فصل پنجم روش پیشنهادی مورد ارزیابی و تحلیل قرار گفته شده است و در نهایت در فصل شش نتایج کلی حاصل از پژوهش به همراه پژوهش‌های آتی بیان شده است.

۱-۱۲- جمع بندی و نتیجه گیری

مسائل بهینه‌سازی طیف وسیعی از مسائل پیرامون ما را در حوزه‌های مختلف در برمی‌گیرند. مسائل بهینه‌سازی واقعی بقدری پیچیده و دشوار می‌باشند که روش‌های کلاسیک مانند گرادیان و سیپملکس قادر به حل مناسب آنها نمی‌باشند. الگوریتم‌های تکاملی نظیر الگوریتم گرده‌افشانی گل‌ها یکی از روش‌های جالب حل اینگونه مسائل می‌باشند. با وجود مزایای الگوریتم‌های تکاملی در یافتن

بهینه اینگونه مسائل، زمان اجرای نسبتاً بالای الگوریتم‌های تکاملی بویژه در مسائل پیچیده و با ابعاد بزرگ می‌تواند از جذابیت استفاده از این روش‌ها را کاهش دهد. در این فصل مقدمات روش پیشنهادی برای تسریع و شتاب الگوریتم گرده‌افشانی گل با استفاده از فناوری موازی‌سازی کودا مورد بررسی قرار گرفته شد.

فصل ۲- ادبیات موضوعی

۲-۱- مقدمه

الگوریتم‌های تکاملی[1] دسته‌ای از روش‌های جستجو و محاسبات تکاملی الهام گرفته از طبیعت می‌باشند. الگوریتم‌های تکاملی از پدیده‌های مختلفی که بیشتر آنها ماهیت زیستی دارند الهام گرفته شده است. یکی از الگوریتم‌های تکاملی، الگوریتم گرده افشانی گل[2] است که بر اساس رفتار گیاهان و تولید نسل آنها به کمک گل مدلسازی شده است. در این فصل در ابتدا الگوریتم گرده‌افشانی گل‌ها شرح داده می‌شود و در ادامه بحث مروری بر معماری موازی واحد پردازش کارت گرافیک[3] خواهیم داشت تا خواننده با این نوع معماری و سبک برنامه نویسی آن آشنا شود. با توجه به اینکه پیاده‌سازی‌های ما بر اساس چارچوب کودا[4] و در محیط برنامه‌نویسی سی‌شارپ[5] توسعه داده شده است در انتهای فصل به شکل مختصر کتابخانه کودا در سی‌شارپ یا کودافای[6] تشریح می‌شود.

[1] Evolutionary algorithms
[2] flower pollination algorithm
[3] Graphics Processing Unit (GPU)
[4] Compute Unified Device Architecture(CUDA)
[5] Csharp or C#.net
[6] Cudafy

۲-۲- الگوریتم گرده افشانی گل

طی میلیون‌ها سال گذشته طبیعت به خوبی توانسته است برای مشکلات پیش‌روی خود راه‌حل‌های مختلفی را ارایه نماید و به نوعی با آن وقف داده شود. سیستم‌های بکار رفته در طبیعت[۱] به خوبی آموخته‌اند که بتواند عملکرد و کارایی خود را در طبیعت پیرامونشان را به شکل بهینه و موثری کار آمد نمایند. زاد و ولد در هر سیستم طبیعی کارآمد یک هدف تکاملی[۲] مهم محسوب می‌شود. سیستم‌های طبیعی به علت موفقیت خوبی که در مواجه با مشکلات پیرامونشان داشتند پایه و اساس الگوریتم‌های الهام گرفته از طبیعت[۳] شده‌اند. از نمونه‌های موفق سابق الگوریتم‌های تکاملی می‌توان به الگوریتم ژنتیک[۴] [۲۱] که بر اساس تکامل و اصول داروین[۵] ارایه شده است و الگوریتم بهینه‌سازی دسته‌جمعی ذرات[۶] که بر اساس رفتار گروهی و دسته جمعی پرندگان و ماهیان ایجاد شده است نام برد[۲۲]. از نمونه‌های جدید الگوریتم‌های تکاملی می‌توان به الگوریتم خفاش[۷] که بر اساس بازخوردهای صوتی و الگوریتم کرم شب‌تاب[۸] بر مبنای الگوهای نورافکنی[۹] این موجودات نیز اشاره نمود[۲۳]. هر کدام از این الگوریتم‌های تکاملی در طیف گسترده‌ای از برنامه‌های کاربردی بکار گرفته می‌شوند. برنامه کاربردی در بسیاری از بخش‌های صنعتی بکار گرفته می‌شود و هدف آن یافتن راه‌حل بهینه[۱۰] برای اینگونه مسائل است که عموماً غیرخطی[۱۱] و چند معیاره[۱۲] محسوب می‌شوند. یافتن راه-حل بهینه در اینگونه مسائل اگر غیرممکن نباشد مسلماً چالش‌برانگیز[۱۳] و دشوار خواهد بود. روش‌های تکاملی به شکل شگفت‌انگیزی در حل مسائل بهینه‌سازی کاربردی بر خلاف روش‌های معمول به خوبی عمل می‌نمایند[۱].

موثر بودن حل مسائل بهینه‌سازی کاربردی به کمک روش‌های تکاملی باعث شده است که در دو دهه اخیر موضوع الگوریتم‌های متاهیوریستیک[۱۴] به شکل قابل توجه‌ای گسترش یابد.

در این بخش یک الگوریتم جدید متاهیوریستیک بر اساس مکانیزم گرده افشانی گل‌ها معرفی و ارایه می‌شود. در این بخش در ابتدا مهمترین ویژگیهای کلیدی در گرده افشانی گل‌ها ارایه شده و سعی

[1] biological systems
[2] evolutionary objectives
[3] nature-inspired algorithms
[4] genetic algorithms
[5] Darwinian
[6] particle swarm optimization
[7] bat algorithm
[8] firefly algorithm
[9] flashing light patterns
[10] optimal solution
[11] nonlinearity
[12] multimodality
[13] challenging
[14] metaheuristics

می‌شود بر اساس این ویژگی‌ها چهار قانون[1] کلیدی جهت مدلسازی الگوریتم پشنهادی ارایه شود. در این بخش الگوریتم گرده‌افشانی گل‌ها مورد بررسی قرار گرفته که هدف نهایی آن بقای نسل‌های شایسته‌تر[2] و از بین رفتن گونه‌های ضعیف‌تر است. در ادامه این بخش با استفاده از توابع ارزیابی[3] شناخته شده الگوریتم مورد نظر مورد ارزیابی و بررسی قرار خواهد گرفت. جهت بررسی دقت و همگرایی گرده‌افشانی گل‌ها از ده تابع ارزیابی و مقایسه با دو الگوریتم ژنتیک[4] و ذرات[5] استفاده شده است. در این بخش الگوریتم تکاملی گرده‌افشانی گل‌ها[6] مورد بررسی قرار خواهد گرفت.

۲-۲-۱- گرده افشانی گلها

حدوده ۲۵۰ هزار گونه گیاهی در طبیعت وجود دارد که چیزی در حدود ۸۰ درصد آنها قادر به تولید گل می‌باشند. از تکامل گلها از دوران کرتاسه[7] تا الان چیزی در حدود ۱۲۵ میلیون سال می‌گذرد به طوریکه تصور جهانی بدون گیاهان گلدار غیر ممکن به نظر می‌رسد. گل را می‌توان اندام جنسی گیاهان گلدار جهت تولید مثل در نظر گرفت. گل‌ها در گرده‌افشانی جهت تکثیر گیاهان نقش مهم و بسزایی ایفاء می‌نمایند. گرده‌افشانی گلها از طریق مختلف نظیر حشرات، پرندگان و سایر حیوانات انجام می‌شود. تعدادی از گیاهان وجود دارند که فقط اجازه گرده‌افشانی را به حشرات و جانوران مشخصی می‌دهند بطوریکه در این گیاهان عمل گرده‌افشانی گلها به شکل تخصصی و پیشرفته صورت می‌پذیرد. حدود ۹۰ درصد گیاهان گلدار برای گرده‌افشانی به وجود جانورانی نظیر پرندگان و حشرات نیاز دارند و تنها ۱۰ درصد آنها نیاز به حشرات یا پرندگان برای گرده‌افشانی ندارند. گیاهانی نظیر بید و چمن جهت گرده‌افشانی فقط به باد و باران متکی می‌باشند و از این نظر گلبرگهای برای جذب حشرات ندارند و اندام گل آنها ساده می‌باشد. گرده‌افشانی حشراتی مانند زنبور عسل گاه فقط مختص به یک گونه گل انجام می‌شود که از این نظر تولید مثل آن گیاه گلدار را تضمین می‌نماید و شانس بقای آن را افزایش می‌دهد. انجام گرده‌افشانی یک گونه گل توسط یک حشره خاص که دارای حافظه اندکی است و فقط قادر به یادگیری محدودی است باعث می‌شود که حشره برای یافتن غذا روی تعداد کمی گل تمرکز نماید و جستجوی کمتری را انجام دهد از طرفی جستجو برای یافتن گیاهانی که شهد مناسبی برای حشره داشته باشند می‌تواند عملی زمان‌بر و پرهزینه برای حشره باشد.

دو نوع گرده‌افشانی خود القایی[8] و دگر القایی[9] در گیاهان گل‌دار دیده می‌شود. در خودالقایی گرده-های[10] یک گل از یک گیاه بر روی یک گل از همان گیاه قرار گرفته می‌شود و در دگر القایی گرده‌های

---

[1] rules
[2] most fittest
[3] Benchmark functions
[4] GA
[5] PSO
[6] flower pollination algorithm (FPA)
[7] Cretaceous period
[8] self-pollination
[9] cross-pollination
[10] pollens

های یک گل از یک گیاه بر روی یک گل از گیاه دیگر قرار گرفته می‌شود. یک نمونه از خودالقایی را می‌توان در گیاه هلو مشاهده کرد که گرده‌های گل‌های آن درخت بر روی گلهای همان درخت قرار گرفته می‌شود. در گرده‌افشانی سراسری(دگرالقایی) گرده‌های گیاهان گلدار توسط حشرات مختلف تا فواصل مختلف پیموده می‌شود[۶].

۲-۲-۲- الگوریتم گرده افشانی گل

با توجه به ویژگی‌های گرده‌افشانی در گیاهان گلدار می‌توانیم چهار قانون ساده را جهت مدلسازی این الگوریتم ارایه نمایم:

۱- گرده‌افشانی از نوع دگر القایی چون از طریق پرواز گرده‌های گل به وسیله حشرات حاصل می‌شود به عنوان گرده‌افشانی سراسری[1] در نظر گرفته می‌شود.
۲- گرده‌افشانی از نوع خودالقایی را حالت گرده‌افشانی محلی[2] در نظر گرفته می‌شود.
۳- شانس بقای یک گل[3] بر حسب تابعی احتمالی از تشابه گلی که با این گل گرده‌افشانی انجام داده بیان می‌شود.
۴- انتخاب نوع گرده افشانی محلی و گرده افشانی سراسری یک گل توسط یک احتمال در بازه $p \in [0,1]$ در نظر گرفته می‌شود.

در دنیای واقعی هر گیاه می‌تواند چندین گل و هر گل می‌تواند تعداد زیادی گرده ایجاد نماید که این فرض مدلسازی مسئله را دشوار می‌نماید لذا جهت سادگی کار می‌توان فرض نمود هر گیاه دارای یک گل و هر گل فقط یک گرده تولید می‌نماید. با این پیش‌فرض می‌توان الگوریتم گرده‌افشانی گل را دو گام اساسی گرده‌افشانی سراسری و محلی ایجاد نمود.

در گام گرده افشانی سراسری گرده هر گل توسط پرواز حشرات به مسافتهای دور برده می‌شود. این نوع گرده‌افشانی باعث می‌شود که گرده بتوانند در محدوده وسیعی از مسئله جابجا شوند. این رفتار گرده‌افشانی سراسری را می‌توان در قالب قانونی که در رابطه(۲-۱) آمده است بیان نمود:

$$x_i^{t+1}=x_i^t+L(x_i^t-g_*) \qquad \textbf{رابطه ۲-۱}$$

در این رابطه X_i^t ، X_i^{t+1}، g_* و L به ترتیب مکان گرده iام در تکرار tام ، مکان گرده iام در تکرار t+1ام، بهترین مکان گرده‌ای که تاکنون پیدا شده است و قدرت گرده‌افشانی نامیده می‌شود که میزان جهش و پرش گرده‌ها را نشان می‌دهد.

[1] global pollination
[2] local pollination
[3] Flower constancy

آنجا که گرده i یا بردار راه حل x_i برای تکرار t و g_* در حال حاضر بهترین راه حل در میان تمام راه حل های نسل فعلی/تکرار است. پارامتر L قدرت گرده افشانی که در اصل اندازه یک گام است. فرض بر این است که قدرت گرده‌افشانی یک عدد مثبت است و آن را به شکل رابطه(۲-۲) نشان می‌دهیم:

رابطه ۲-۲

$$L \sim \frac{\lambda \Gamma(\lambda)\sin(\frac{\pi\lambda}{2})}{\pi} \times \frac{1}{s^{1+\lambda}} \ , s>>s_0 > 0$$

در این رابطه $\Gamma(\lambda)$ را تابع گامای استاندارد[1] می‌نامند و مقدار مناسب برای این تابع $\lambda = 1.5$ انتخاب می‌شود.

گرده افشانی محلی یا همان خودالقایی گلها را می‌توان طبق رابطه(۲-۳) مدلسازی نمود[۶]:

رابطه ۲-۳

$$X_i^{t+1} = X_i^t + \varepsilon(X_j^t - X_k^t)$$

در این رابطه، x^t_j و x^t_k دو گرده مختلفی است که در گل‌های مشابه ایجاد شده‌اند. در شکل(۲-۱) شبه کد الگوریتم گرده‌افشانی گلهابا اعمال ۴ قانون اصلی آن نشان داده شده است. همانگونه که در این شبه کد مشخص شده است یک عدد تصادفی بین ۰ تا ۱ ایجاد می‌شود و اگر این عدد کمتر از ۰.۵ باشد گرده افشانی سراسری و اگر بیشتر از این مقدار باشد گرده افشانی از نوع محلی انجام می‌شود. در واقع عدد تصادفی میزان وقوع گرده‌افشانی سراسری یا محلی را کنترل می‌نماید[۶].

Flower Pollination Algorithm (or simply Flower Algorithm)

Objective min *or* max $f(\mathbf{x})$, $\mathbf{x} = (x_1, x_2, ..., x_d)$
Initialize a population of n flowers/pollen gametes with random solutions
Find the best solution $\mathbf{g}_*$ *in the initial population*
Define a switch probability $p \in [0, 1]$
while *(t <MaxGeneration)*
  for $i = 1 : n$ *(all n flowers in the population)*
    if *rand* $< p$,
      Draw a (d-dimensional) step vector L which obeys a Lévy distribution
      Global pollination via $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + L(\mathbf{g}_* - \mathbf{x}_i^t)$
    else
      Draw ϵ *from a uniform distribution in [0,1]*
      Randomly choose j and k among all the solutions
      Do local pollination via $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \epsilon(\mathbf{x}_j^t - \mathbf{x}_k^t)$
    end if
    Evaluate new solutions

شکل ۲-۱: شبه کد الگوریتم گرده افشانی گل [۶]

[1] standard gamma

شبیه‌سازی و آزمایشات مختلف در تعداد زیادی مسئله بهینه‌سازی کاربردی نشان می‌دهد که بهترین مقدار p در شبه کد بالا عددی در حدود ۰.۸ می‌باشد.

۲-۲-۳- ارزیابی گرده افشانی گل

هر الگوریتم تکاملی جدید باید از نظر دقت و همگرایی با سایر الگوریتم‌های تکاملی مورد مقایسه قرار گیرد. برای ارزیابی همگرایی و دقت الگوریتم‌های تکاملی از توابع آزمون[۱] یا ارزیابی که تعداد آنها می-تواند صدها مورد باشد استفاده می‌نمایم. در این پژوهش از ده تابع آزمون مشهور که در جدول(۲-۱) آورده شده‌اند برای تست همگرایی و دقت الگوریتم گرده‌افشانی گلها استفاده می‌شود.

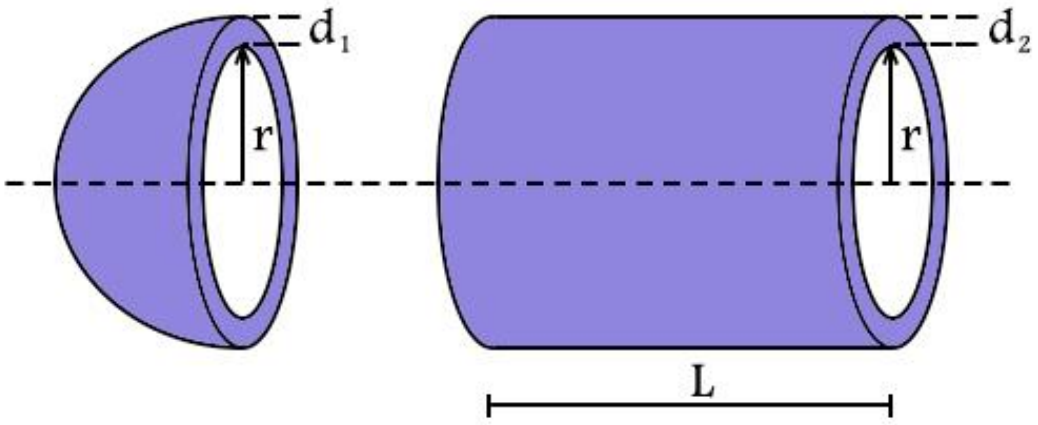
xing برای سنجش همگرایی الگوریتم گرده‌افشانی گلها[۲] آن را با الگوریتمهای ژنتیک[۳] و ذرات[۴] مورد مقایسه قرار داد و شرط پایان الگوریتمها را خطای آستانه 10^{-5} در نظر گرفت. در این آزمایش حداکثر تعداد آزمایش الگوریتمها ۱۰۰ و تعداد جمعیت اولیه[۵] را n=25 و ضریب احتمالی گرده‌افشانی سراسری و محلی p=0.8 و برای الگوریتم ژنتیک نرخ ترکیب[۶] ۰.۹۵ و احتمال جهش[۷] نیز ۰.۵ تنظیم شد و برای الگوریتم ذرات پارامتر یادگیری[۸] c1=c2=2 انتخاب شده است. در جدول مورد نظر یک مقایسه بر اساس ۱۰ تابع ارزیابی بین الگوریتمهای گرده‌افشانی گل، ذرات و ژنتیک صورت گرفته است. در این جدول عددی مانند $3341 \pm 649(100\%)$ نشان می‌دهد که متوسط تعداد تکرار الگوریتم گرده‌افشانی گلها ۳۳۴۱ با انحراف معیاری نزدیک عدد ۶۴۹ با دقت ۱۰۰ درصدی و با میزان خطای کمتر از 10^{-5} به جواب مسئله رسید. اطلاعات جدول نشان می‌دهد که در تکرارهای کمتر نسبت به الگوریتم ژنتیک و ذرات الگوریتم گرده‌افشانی گلها به جوابهای مسئله با دقت بیشتری همگرا می‌شود[۶]:

[1] test functions
[2] FPA
[3] GA
[4] PSO
[5] population size
[6] crossover
[7] mutation probability
[8] learning parameters

جدول ۲-۱: مقایسه دقت الگوریتم گرده افشانی گل، ژنتیک و ذرات [۶]

Functions/Algorithms	GA	PSO	FPA
Michalewicz (d=16)	$89325 \pm 7914(95\%)$	$6922 \pm 537(98\%)$	$3341 \pm 649(100\%)$
Rosenbrock (d=16)	$55723 \pm 8901(90\%)$	$32756 \pm 5325(98\%)$	$5532 \pm 1464(100\%)$
De Jong (d=256)	$25412 \pm 1237(100\%)$	$17040 \pm 1123(100\%)$	$4245 \pm 545(100\%)$
Schwefel (d=128)	$227329 \pm 7572(95\%)$	$14522 \pm 1275(97\%)$	$6851 \pm 448(100\%)$
Ackley (d=128)	$32720 \pm 3327(90\%)$	$23407 \pm 4325(92\%)$	$3357 \pm 968(100\%)$
Rastrigin	$110523 \pm 5199(77\%)$	$79491 \pm 3715(90\%)$	$10840 \pm 2689(100\%)$
Easom	$19239 \pm 3307(92\%)$	$17273 \pm 2929(90\%)$	$4017 \pm 982(100\%)$
Griewank	$70925 \pm 7652(90\%)$	$55970 \pm 4223(92\%)$	$4918 \pm 1429(100\%)$
Yang ($d = 16$)	$27923 \pm 3025(83\%)$	$14116 \pm 2949(90\%)$	$4254 \pm 1839(100\%)$
Shubert(18 minima)	$54077 \pm 4997(89\%)$	$23992 \pm 3755(92\%)$	$9271 \pm 1758(100\%)$

جهت بررسی میزان همگرایی الگوریتم گرده‌افشانی گلها، ژنتیک و ذرات به بررسی یک مثال کاربردی و حل آن به وسیله این الگوریتمها می‌پردازیم. مخازن تحت فشار را می‌توان در بسیاری از کاربردها و حتی در طراحی بطری‌های نوشیدنی یافت. هدف اصلی از طراحی بهینه مخازن فشار به حداقل ساندن هزینه‌ها با توان مقابله با مقداری مشخص از فشار است. در شکل(۲-۲)، نمایشی از یک مخزن ذخیره گاز و مشخصه‌های مهم آن نشان داده شده است[۱].



شکل ۲-۲: طراحی بهینه یک مخزن ذخیره گاز

متغیرهایی مهمی که در طراحی این کپسول فشار بکار می‌روند عبارتند ازd_1 ضخامت کلاهک، d_2 ضخامت بدنه، r شعاع داخلی و L طول بخش استوانه‌ای می‌باشد. طراحی بهینه کپسول فشار در واقع یک مسئله بهینه‌سازی محسوب می‌شود که رابطه(۲-۴) معادله آن را نشان می‌دهد.

$$\text{Minimize} f(x) = 0.6224 d_1 r L + 1.7781 d_2 r^2 + 3.1661 d_1^2 L + 19.84 d_1^2 r \qquad \textbf{رابطه ۲-۴}$$

رابطه(۲-۵) تا (۲-۸) شرایط و مرزهای این مسئله بهینه‌سازی را نشان می‌دهد. این مسئله بهینه-سازی دارای چهار شرط خطی می‌باشد[۶].

$$g_1(x) = -d_1 + 0.0193r \leq 0$$ **رابطه ۲-۵**

$$g_2(x) = -d_2 + 0.00954r \leq 0$$ **رابطه ۲-۶**

$$g_3(x) = -\pi r^2 L - \frac{4\pi}{3} r^3 + 1296000 \leq 0$$ **رابطه ۲-۷**

$$g_4(x) = L - 240 \leq 0$$ **رابطه ۲-۸**

متغیرهای از d_1 ضخامت کلاهک، d_2 ضخامت بدنه دارای محدودهای مشخص شده در رابطه(۲-۹) است.

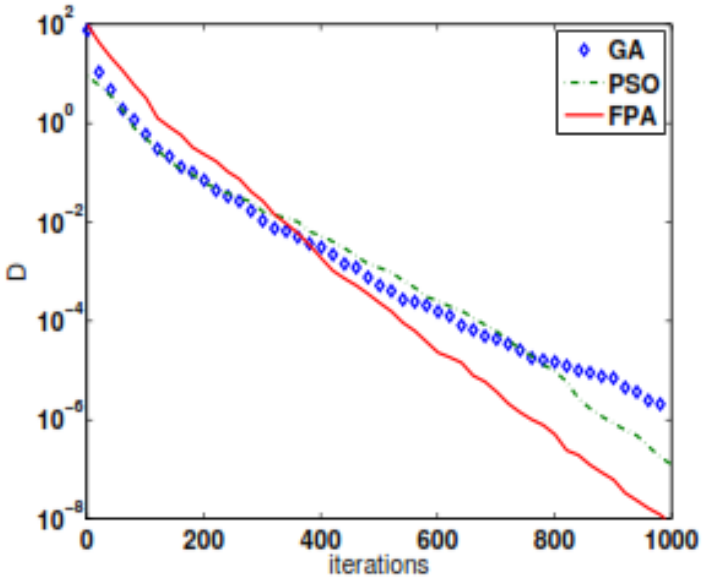
$$0.0625 \leq d_1, d_2 \leq 99 \times 0.0625$$ **رابطه ۲-۹**

متغیرهای r شعاع داخلی و L طول بخش استوانه‌ای دارای محدودهای مشخص شده در رابطه(۲-۱۰) است:

$$10.0 \leq r$$ **رابطه ۲-۱۰**

کمترین مقدار تابع هدف در رابطه(۲-۱۱) که توسط الگوریتم گرده‌افشانی گل بدست آمده F^* ≈ 6059.714 و آن هم به ازای مقادیر $X^* \approx (0.8125, 0.4357, 42.0984, 176.6366)$ که به ترتیب به مقادیر از d_1 ضخامت کلاهک، d_2 ضخامت بدنه، r شعاع داخلی و L طول بخش استوانه‌ای اشاره دارد. مقادیر بدست آمده برای تعداد تکرار ۴۰ می‌باشد که با افزایش تکرار میزان خطا مطابق شکل(۲-۳) در هر سه نمودار گرده‌افشانی گل[1]، ژنتیک[2] و ذرات[3] کاهش می‌یابد[۶].

[1] FPA
[2] GA
[3] PSO



شکل ۲-۳: مقایسه دقت و همگرایی الگوریتم گرده‌افشانی گلها، ژنتیک و ذرات [۶]

بررسی الگوریتم گرده‌افشانی گل نشان داد که این الگوریتم نرخ همگرایی بهتری نسبت به الگوریتم-های ذرات و ژنتیک دارد و توانایی حل مسائل بهینه را به شکل دقیق دارا می‌باشد. چالش الگوریتم تکاملی گرده‌افشانی گلها مانند هر الگوریتم تکاملی زمان اجرای قابل توجه در حل مسائل بهینه‌سازی پیچیده‌تر می‌باشد که یکی از روش‌های کاربردی برای کاهش زمان اجراء استفاده از معماری موازی و روش‌های موازی‌سازی مختلف است. در فصل چهارم برای کاهش زمان اجرای الگوریتم گرده‌افشانی گلها یک نسخه موازی به کمک فناوری کودا ارایه خواهد شد.

۲-۲-۴- تفاوت الگوریتم گرده افشانی گل با الگوریتم های نقطه ای مانند الگوریتم ژنتیک

الگوریتم های تکاملی از دیدگاه استراتژی نزدیک شدن به جواب بهینه، به دو دسته منطقه محور[1] و نقطه محور[2] تقسیم بندی می شوند که الگوریتم گرده افشانی گل از نوع منطقه محور می باشد.
در برخی موارد (بسته به نوع مسئله) الگوریتم منطقه محور بر نقطه محور برتری دارد. به عنوان مثال در بخش ۲-۲-۳ و محاسبه پارامترهای بهینه ساخت کپسول تحت فشار، از آنجایی که در هر تکرار الگوریتم های منطقه محور، اعضای جمعیت اولیه به سمت عضو شایسته تر حرکت می کنند، بنابراین به تدریج تجمع اعضای جمعیت حول جواب بهینه بیشتر می شود و در صورت بروز خطا در الگوریتم، حداقل جوابی نزدیک به جواب بهینه تولید می شود. در حالی که در الگوریتم های نقطه محور،

[1] region-based
[2] point-based

استراتژی نزدیک شدن اعضای جمعیت به سمت عضو بهینه وجود ندارد. و در صورت بروز خطا ممکن است جواب پرت تری نسبت به جواب بهینه تولید شود.
بنابراین در صورت بروز خطای زیاد در هر دو الگوریتم گرده افشانی گل و ژنتیک، استفاده از جواب الگوریتم ژنتیک ممکن است منجر به تولید کپسول تحت فشاری کاملا غیر ایمن گردد، در صورتی که تولید کپسول تحت فشار با جواب الگوریتم گرده افشانی گل به احتمال بسیار بسیار کمتری منجر به تولید کپسول تحت فشار غیر ایمن می گردد. به بیان ساده تر در برخی موارد جوابهای تولید شده توسط الگوریتم های ناحیه محور قابل اتکاتر هستند.

۲-۳- پردازش موازی

در این قسمت نحوه موازی‌سازی الگوریتم‌ها و معماری موازی و غیرموازی پردازنده تشریح شده و سعی می‌شود که بخش زیادی از مطالب این فصل به پردازش موازی به کمک پردازنده گرافیکی و چارچوب کودا پرداخته شود.

۲-۳-۱- ساختار پردازنده

طی دهه‌های گذشته تمایل برای انجام محاسبات به شکل موازی[۱] روز به روز در حال گسترش و توسعه بوده است. هدف نهایی و اصلی در پردازش موازی بهبود سرعت محاسبات و کاهش زمان اجرای برنامه‌ها می‌باشد. تعاریف مختلفی برای موازی‌سازی ارایه شده است اما از دیدگاه محاسباتی، اجرای قطعات کوچک برنامه‌های بزرگ به شکل مستقل و مجزاء از هم تعریف می‌شود. اجرای این قطعات ریزتر به شکل موازی در منابع محاسباتی[۲] که می‌تواند کامپیوتر یا هسته‌های پردازشی[۳] باشد انجام می‌گیرد.

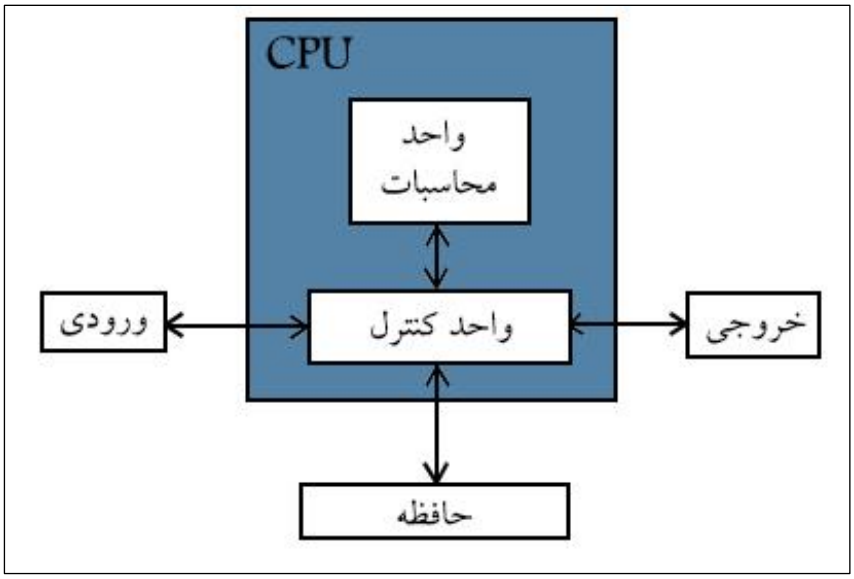
جهت انجام پردازش موازی معماری و نرم‌افزار بکار رفته دو جزء کلیدی محسوب می‌شوند. برای هر پردازش موازی دو مولفه کلیدی ذیل وجود دارد:

- معماری کامپیوتر[۴] به عنوان جزء سخت‌افزاری
- برنامه‌نویسی موازی[۵] به عنوان جزء نرم‌افزاری

[1] parallel computation
[2] computing resources
[3] cores
[4] Computer architecture
[5] Parallel programming

معماری کامپیوتر به عنوان جزء سخت‌افزاری در موازی‌سازی نقش مهمی را ایفاء می‌نماید در شکل(۲-۴) معماری پردازنده‌های امروزی با اجزاء آن نشان داده شده است. در پردازنده‌های امروزی سه جزء مهم وجود دارد[۱۰]:

- حافظه[۱]
- واحد کنترل و محاسبات[۲]
- ورودی و خروجی[۳]



شکل ۲-۴: معماری و اجزاء پردازنده‌های امروزی

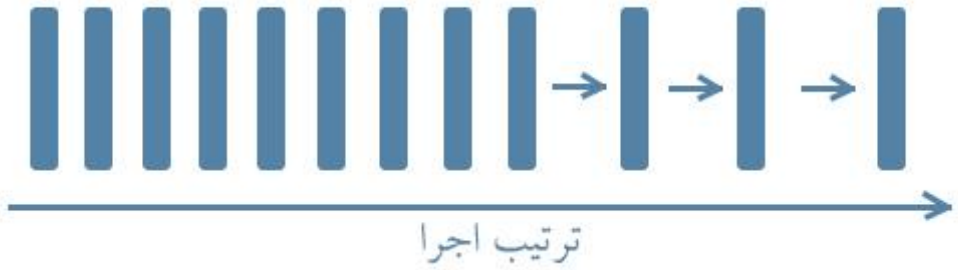
واحد کنترل و پردازش منطقی نقش مهمی در کارایی پردازنده اصلی دارد. کارایی این واحد به اجزای کوچکتری به نام هسته[۴] محاسباتی بستگی دارد. تعداد هسته‌ها در کامپیوترهای سابق به یک عدد محدود می‌شدند و به آنها تک هسته‌ای[۵] می‌گفتند و این در حالی است که پردازنده‌های امروزی تعداد بسیار بیشتری هسته محاسباتی دارند و به آنها پردازنده‌های چندهسته‌ای[۶] گفته می‌شود. یک برنامه برای اجرای موازی باید به درستی بتواند اجزاء خود را به هسته‌های پردازنده نگاشت نماید و در این حالت برنامه‌نویس نیاز است تا حدود زیادی با معماری سخت‌افزاری پردازنده آشنایی داشته باشد که این شرط آشنایی در حالت برنامه‌نویسی غیرموازی و سریال[۷] الزامی نمی‌باشد. به طور کلی می‌توان

[1] Memory
[2] Central processing unit
[3] Input/Output interfaces
[4] core
[5] uniprocessor
[6] multicore
[7] Serial

گفت که در انواع مختلف پردازش موازی برنامه‌نویس نیاز است با معماری موازی سخت‌افزار[1] تا حدود زیادی آشنایی داشته باشد[۱۰].

۲-۳-۲- اجرای موازی و غیر موازی

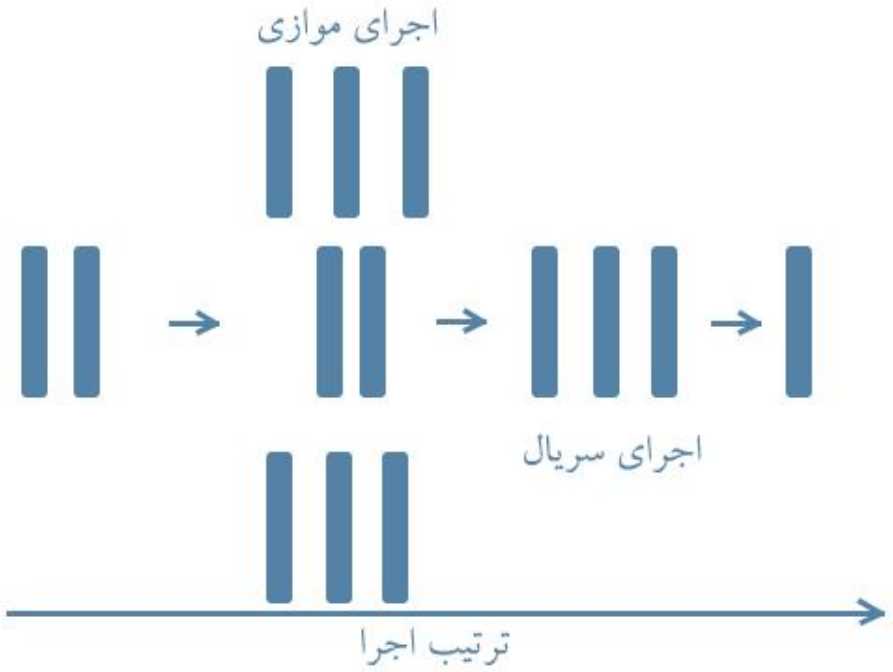
برنامه‌های کامپیوتری جهت اجراء در پردازنده نیاز است به اجزای کوچکتری تقسیم شده و سپس اجراء شوند. بسته به ترتیب اجرای این اجزاء دو حالت کلی سریال یا موازی قابل تصور است. به عنوان مثال در شکل(۲-۵)، اجزاء برنامه با ترتیب و پشت سرهم اجراء می‌شوند به گونه‌ای که برای اجراء در پردازنده سیستم که تعداد هسته‌های آن محدود می‌باشد ناگزیر به ایجاد صف اجراء می‌شوند.



شکل ۲-۵: اجرای سریال برنامه‌ها

در شیوه موازی برنامه یا کارها، در یک لحظه زمانی تعدادی پردازش موازی با هم انجام گرفته می‌شود ولی سطح توازی سازی برنامه به ماهیت آن بستگی دارد به عنوان مثال در شکل(۲-۶)، قسمتی از برنامه که دارای قابلیت موازی بوده به شکل موازی اجراء شده و قسمتی دیگر از برنامه که دارای سطح موازی‌سازی مناسب نبوده به شکل سریال به اجراء گذاشته شده است. البته گاهی اوقات محدودیت-های سخت‌افزاری باعث می‌شود که فقط قسمتی از برنامه روند موازی را طی نماید و مابقی برنامه به شکل سریال اجراء شود[۱۰]:

[1] multicore architectures.



شکل ۲-۶: اجرای موازی برنامه‌ها

۲-۳-۳- انواع موازی سازی

امروزه دو نوع موازی‌سازی رایج کاربردی در اکثر برنامه‌های موازی به شرح ذیل وجود دارد[۱۰]:

- موازی‌سازی وظایف[۱]
- موازی‌سازی داده‌ها[۲]

موازی‌سازی وظایف، زمانی استفاده می‌شود که تعداد زیادی وظیفه و تابع می‌توانند به شکل مستقل و موازی اجراء شوند. در این شیوه موازی‌سازی، توابع و وظایف به شکل موازی در سیستم‌های چندپردازنده‌ای اجراء می‌شوند.

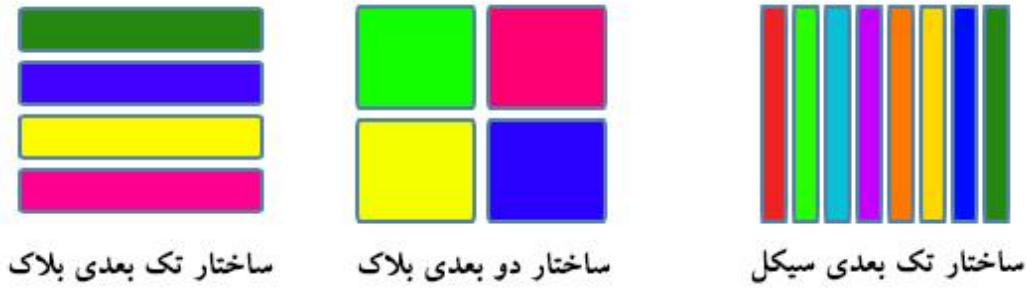
موازی‌سازی داده‌ها، زمانی استفاده می‌شود که تعداد زیادی داده می‌توانند به شکل مستقل و موازی اجراء شوند. در این شیوه موازی‌سازی داده‌ها به شکل موازی در سیستم‌های چندپردازنده‌ای به شکل موازی اجراء می‌شوند. برنامه نویسی کودا[۳] یکی از چارچوبهای بسیار مناسب برای پردازش‌های از نوع موازی‌سازی داده‌ها محسوب می‌شود. در این سبک موازی‌سازی مجموعه‌ای از نخ‌ها[۴] به موازی سازی داده‌ها می‌پردازند به گونه‌ای که هر نخ بخشی از داده‌ها را مدیریت می‌نمایند. در تقسیم‌بندی داده‌ها دو شیوه متداول تقسیم‌بندی بلاک[۵] و تقسیم‌بندی سیکل[۶] به چشم می‌خورد. در روش بلاک‌بندی هر

[1] Task parallelism
[2] Data parallelism
[3] CUDA programming
[4] Threads
[5] block partitioning
[6] cyclic partitioning

تکه[1] از داده‌ها توسط یک نخ مدیریت می‌شوند و در حالت تقسیم‌بندی سیکل تعدادی از تکه‌ها با یک نخ مدیریت می‌شود. در شکل(۲-۷) این دو حالت تقسیم‌بندی داده‌ها برای موازی‌سازی نشان داده شده است.

شکل ۲-۷: تقسیم‌بندی داده‌ها به دو شیوه بلاک و سیکل در موازی سازی کودا

ساختار قرار گیری بلاک می‌تواند به شکل یک بعدی، دو بعدی و حتی سه بعدی باشد. به عنوان نمونه در شکل(۲-۸) و تصویر سمت چپ و وسط به ترتیب بلاک‌ها در ساختار تک بعدی[2] و دوبعدی سازماندهی شده و در تصویر سمت راست تقسیم‌بندی مبتنی بر سیکل در حالت تک بعدی نمایش داده شده است[۱۰]:



شکل ۲-۸: تقسیم‌بندی داده‌ها به دو شیوه بلاک و سیکل در موازی سازی کودا

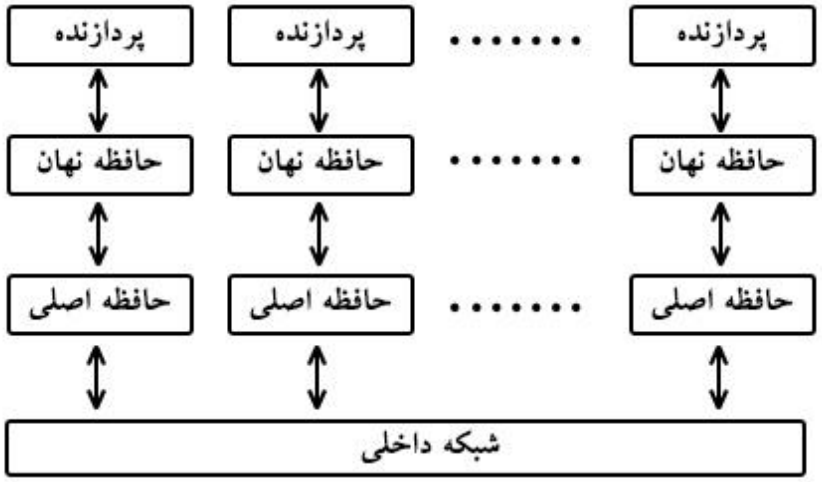
۲-۳-۴- معماری موازی سازی در کامپیوتر

به طور کلی چهار معماری مختلف برای کامپیوترهای امروزی به شرح ذیل وجود دارد.

[1] chunk
[2] one-dimensionally

- تک دستورالعمل و تک داده[1]: این معماری بیشتر در کامپیوترهای قدیمی وجود دارد که مدل سریالی نیز نامیده می‌شود و پردازنده آنها دارای یک هسته است.
- تک دستورالعمل و چند داده[2]: این مدل معماری مختص موازی سازی در سیستم‌های چند پردازنده می‌باشد که یک دستورالعمل مشخص بر روی تعداد زیادی داده انجام می‌شود.
- چند دستورالعملی و تک داده[3]: این معماری یک معماری غیر مرسوم است.
- چند دستورالعملی و چند داده‌ای[4]: این معماری موازی جهت محاسبات خود از معماری‌های مختلف تک دستورالعمل و تک داده به عنوان زیر سیستم استفاده می‌نماید.

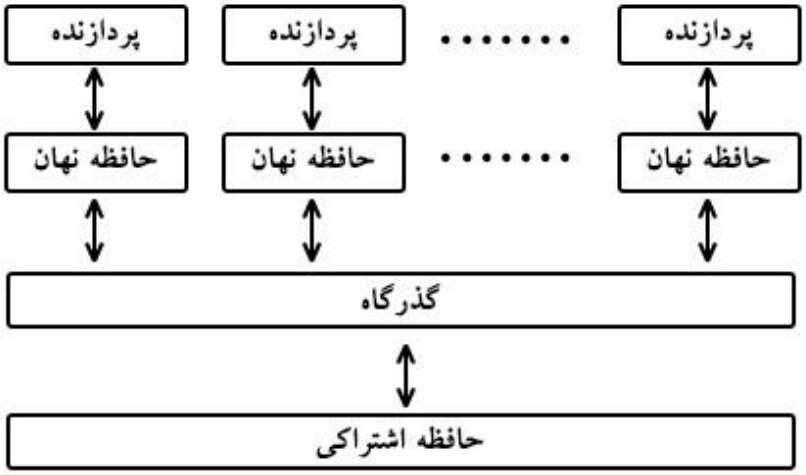
سیستم‌های موازی‌سازی به طور کلی دارای دو معماری عمده سیستم‌های چند گره‌ای[5] و چند پردازنده‌ای می‌باشند. در سیستم‌های چند گره‌ای، پردازنده‌های سیستم‌های مختلف که گره‌های یک شبکه محسوب می‌شوند یک معماری موازی مانند شکل(۲-۱۰) ایجاد می‌نمایند[۱۰]:



شکل ۲-۹: معماری موازی بر اساس گره‌های شبکه

در سیستم‌های چند پردازنده‌ای، پردازنده‌های یک سیستم یک معماری موازی مانند شکل(۲-۱۱) ایجاد می‌نمایند:

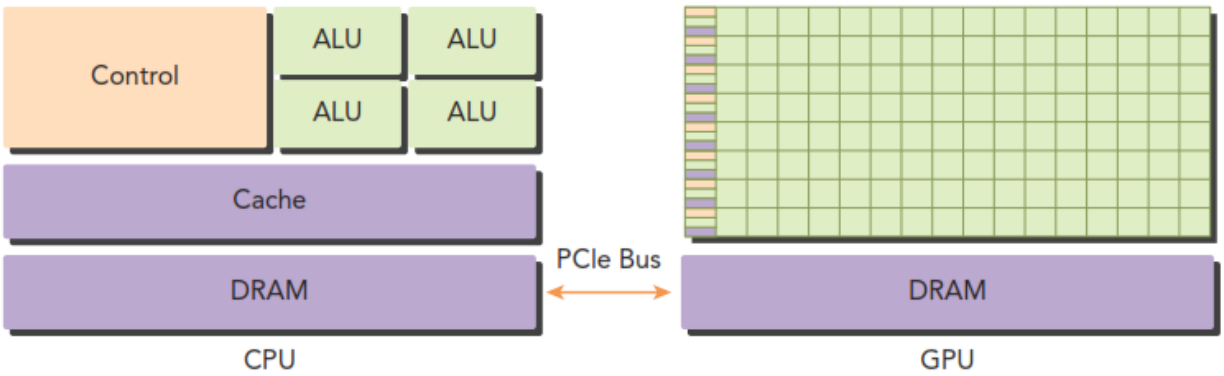
1 Single Instruction Single Data (SISD)
2 Single Instruction Multiple Data (SIMD)
3 Multiple Instruction Single Data (MISD)
4 Multiple Instruction Multiple Data (MIMD)
5 multi-node system



شکل ۲-۱۰: معماری موازی بر اساس سیستم چند پردازنده‌ای

۲-۳-۵- واحد پردازش کارت گرافیک

واحد پردازش کارت گرافیک را می‌توان پردازنده کارت گرافیک برای انجام عملیات گرافیکی و رندر کردن تصاویر در نظر گرفت. تعداد هسته‌های موجود در پردازنده گرافیکی به مراتب بیشتر از پردانده اصلی است و تعداد هسته‌ها گاهی تا صدها و هزاران عدد می‌رسد. معماری پردازنده گرافیکی به صورتی است که بیشتر معماری آن صرف طراحی واحد پردازش و محاسبات[1] آن شده و تمرکز کمتری بر طراحی حافظه اصلی[2] و پنهان[3] آن صورت گرفته است چون پردازش‌های که در پردازنده گرافیکی انجام می‌شود بیشتر ازنوع پردازش‌های ساده ولی با حجم بالا می‌باشد، نیاز است قسمت محاسبات و منطق گسترش بیشتری نسبت به سایر بخش‌ها داشته باشد. در شکل(۲-۱۲) تفاوت معماری سخت-افزاری پردازنده گرافیکی و اصلی به تصویر کشیده شده است[۱۰]:



شکل ۲-۱۱: تفاوت معماری پردازنده گرافیکی و اصلی [۱۰]

1 ALU
2 DRAM
3 Cash

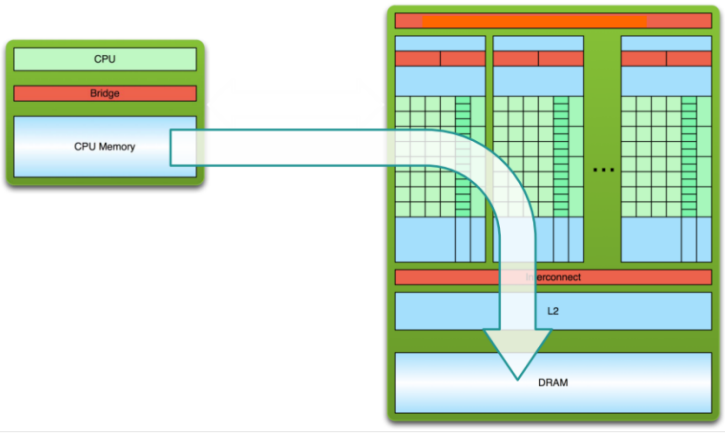
همانگونه که در شکل(۲-۱۲) نشان داده شده است پردازنده گرافیکی به جای ذخیره کردن اطلاعات در حافظه موقت یا پنهان سعی می‌نماید بر روی داده‌ها عملیات پردازشی انجام دهد پس نیاز است که مدارات آن بیشتر برای محاسبات طراحی شوند و کمتر از این مدارات برای ذخیره اطلاعات استفاده نماید و این در حالی است که در پردازنده اصلی بیشتر مدارات برای طراحی واحد ذخیره اطلاعات پنهان، حافظه، واحد کنترل و غیره تدارک دیده شده است. بررسی شکل بالا نشان می‌دهد که در بخش بیشتری از پردازنده گرافیکی نسبت به پردازنده اصلی مدارات محاسبات ایجاد شده‌اند. پردازنده گرافیکی برای ارتباط برقرار کردن با پردازنده اصلی از یک گذرگاه استفاده می‌نماید که یک پل ارتباطی بین پردازنده گرافیکی و اصلی جهت تبادل داده‌ها می‌باشد. در برنامه نویسی موازی پردازنده گرافیکی، نیاز است که کدهای نوشته شود در هر دو سمت پردازنده اصلی و گرافیکی قابل اجراء باشند به این دلیل، در برنامه نویسی پردازنده گرافیکی از دو نوع کد ذیل استفاده می‌نماییم.

- کد میزبان[1](پردازنده اصلی)
- کد دستگاه[2](پردازنده گرافیکی)

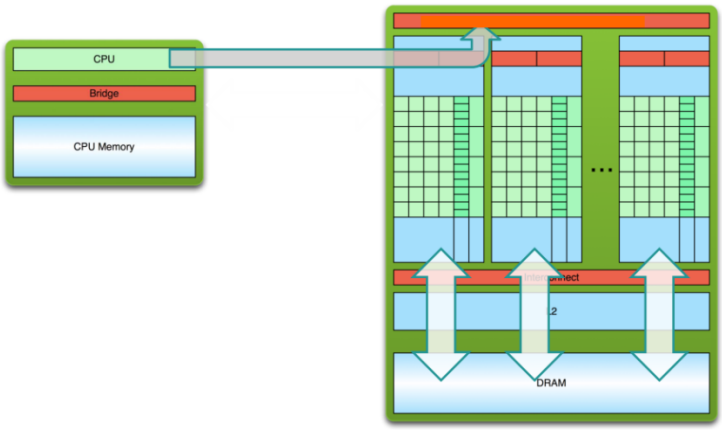
کد برنامه موازی به گونه‌ای است که در ابتدا در پردازنده اصلی اجراء شده و بخش‌های از آن کد در معماری دستگاه به اجراء گذاشته می‌شود.

برای اجرای یک برنامه موازی در واحد پردازش کارت گرافیک نیاز است که جریانی از داده‌ها بین پردازنده گرافیکی و اصلی جابجا شود. فرآیند جریان اطلاعات بین پردازنده گرافیکی و اصلی را می‌توان در سه مرحله اصلی نشان داد. در مرحله اول انتقال داده‌ها از حافظه پردازنده اصلی به حافظه پردازنده گرافیکی مطابق شکل(۲-۱۴) صورت می‌پذیرد و در مرحله دوم پردازنده اصلی تابعی موسوم به کرنل[3] را که کدهای موازی سازی درون آن قرار دارد را به پردازنده گرافیکی تحویل داده تا آن را اجراء نماید(شکل(۲-۱۵)) و در نهایت پردازنده گرافیکی بعد از اتمام مراحل پردازش موازی داده‌های خروجی مورد نظر را به حافظه اصلی پردازنده گرافیکی ارسال می‌نماید(شکل(۲-۱۶))[۹]:

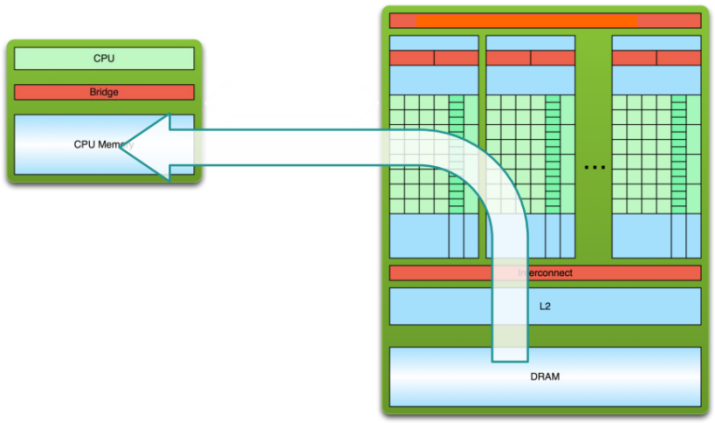
1 Host Code
2 Device Code
3 Kernel



شکل ۲-۱۲: انتقال اطلاعات از حافظه پردازنده اصلی به حافظه پردازنده گرافیکی [۹]



شکل ۲-۱۳: انتقال اطلاعات از پردازنده اصلی به پردازنده گرافیکی و اجرای کرنل برنامه موازی توسط پردازنده گرافیکی[۹]



شکل ۲-۱۴: انتقال اطلاعات از حافظه پردازنده گرافیکی به حافظه پردازنده اصلی [۹]

۲-۳-۵-۱- توان محاسباتی کارتهای گرافیکی

کارت‌های گرافیکی شرکت انویدیا[1] که دارای پردازنده‌های با توان محاسباتی موازی می‌باشند به چهار دسته ذیل تقسیم‌بندی شده‌اند[۱۰]:

- تیگرا[2]: این نوع پردازنده گرافیکی عرضه شده توسط شرکت انویدیا بیشتر بر روی وسایلی نظیر تبلت و گوشی‌های موبایل استفاده می‌شود.
- جیفورس[3]: این پردازنده گرافیکی بیشتر در کاربردهای خانگی و سیستم‌های معمولی عرضه می‌شود.
- کوآردو[4]: یک پردازنده حرفه‌ای در تهیه تصاویر گرافیکی محسوب می‌شود و بیشتر در کاربردهای از نوع بصری‌سازی حرفه‌ای[5] از آن استفاده می‌شود.
- تسلا[6]: این پردازنده گرافیکی در دیتاسنترها[7] و پردازش‌های موازی آنها بطور خاص کاربرد دارد.

قدرت محاسباتی هر کدام از این نوع پردازندها به عوامل مختلفی بستگی دارد که دو عامل زیر تا حد زیادی میزان قدرت پردازشی و گرافیکی آنها را تعیین می‌نماید:

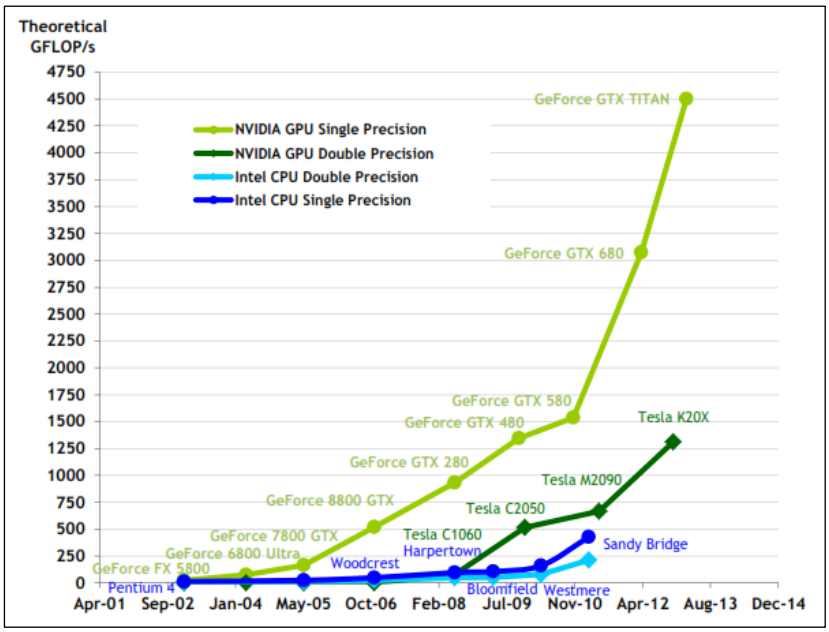
- تعداد هسته‌های کودا[8]
- میزان حافظه[9]

البته دو معیارهای بالا مهمترین متریک‌های سنجش قدرت یک کارت گرافیک محسوب می‌شوند، با این وجود معیارهای ذیل نیز نقش مهمی در عملکرد و کارایی پردازندهای کارت گرافیک دارند[۱۰]:

- عملکرد محاسباتی[10]
- پهنای باند حافظه[11]

عملکرد محاسباتی در واقع تعداد محاسبات ممیز اعشار است که یک پردازنده گرافیکی می‌تواند در یک ثانیه انجام دهد. در شکل(۲-۱۷)، سیر تکاملی و افزایش محاسبات ممیز اعشار در پردازنده گرافیکی و اصلی را نشان داده است. همانگونه که در این شکل مشخص شده است تعداد محاسبات ممیز اعشار در پردازنده‌های گرافیکی نسبت به پردازنده‌های اصلی رشد بیشتری داشته است:

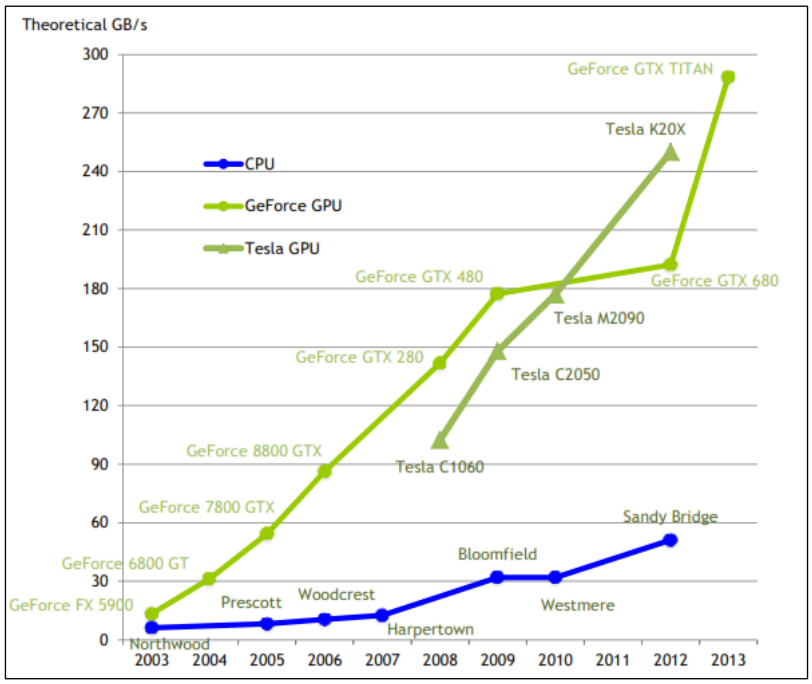
1 NVIDIA
2 Tegra
3 Geforce
4 Quadro
5 professional visualization
6 Tesla
7 datacenter
8 Number of CUDA cores
9 Memory size
10 Peak computational performance
11 Memory bandwidth



شکل ۲-۱۵: مقایسه تعداد محاسبات ممیز اعشار در پردازنده گرافیکی و اصلی [۲۴]

در کنار عملکرد محاسباتی که با تعداد محاسبات ممیز شناور در واحد زمان سنجیده می‌شود توانایی گذرگاه‌های ارتباطی بین پردازنده اصلی و گرافیکی نیز یکی از عوامل مهم و تاثیر گذار در افزایش کارایی پردازنده‌های گرافیکی محسوب می‌شود. در شکل(۲-۱۸)، سیر افزایش پهنای باند محاسبات در کارت‌های گرافیکی جیفورس[۱]، تسلاء[۲] و پردازنده اینتل[۳] در یک بازه زمانی از سال ۲۰۰۳ تا ۲۰۱۳ را نشان داده است. همانگونه که در این شکل مشخص شده است پهنای باند محاسبات[۴] در پردازنده‌های گرافیکی نسبت به پردازنده‌های اصلی رشد بیشتری داشته است[۲۴]:

1 Geforce
2 Tesla
3 Intel
4 Bandwidth Calculations



شکل ۲-۱۶: مقایسه پهنای باند محاسبات در پردازنده گرافیکی و اصلی [۲۴]

به عنوان نمونه در جدول(۲-۲) دو پردازنده گرافیکی FERMI و KEPLER با چهار ویژگی تعداد هسته‌های کودا، میزان حافظه، عملکرد محاسباتی و پهنای باند حافظه مورد مقایسه و ارزیابی قرار گرفته شده است. پردازنده گرافیکی FERMI وKEPLER دو پردازنده از خانواده پردازنده‌های تسلا می‌باشند که به ترتیب در سال ۲۰۱۰ و ۲۰۱۲ عرضه شده‌اند. مقایسه اطلاعات این جدول نشان می-دهد که کارایی و قدرت پردازنده KEPLER بیشتر از پردازنده FERMI است:

جدول ۲-۲: مقایسه کارایی و عملکرد پردازنده گرافیکی FERMI و KEPLER

FERMI (TESALA C2050)	KEPLER (TESLA K10)	
448	2×1536	تعداد هسته
6 GB	8 GB	حافظه
1.03 Tflops	4.58 Tflops	حداکثر توان محاسباتی
144 GB/s	320 GB/s	پهنای باند

شرکت انویدیا جهت مشخص نمودن توان محاسباتی پردازنده‌های خود از واژه ظرفیت محاسباتی[1] استفاده می‌نماید که نسخه سخت‌افزار و معماری پردازنده گرافیکی را نشان می‌دهد. در جدول(۲-۳)، یک مقایسه بین ظرفیت محاسباتی چند پردازنده گرافیکی از خانواده تسلا نشان داده شده است. هر

1 compute capability

چقدر مقدار ظرفیت محاسباتی یک پردازنده کارت گرافیک بیشتر باشد کارایی آن بالاتر خواهد بود[۱۰]:

جدول ۲-۳: مقایسه ظرفیت محاسباتی چند پردازنده گرافیکی از خانواده تسلا

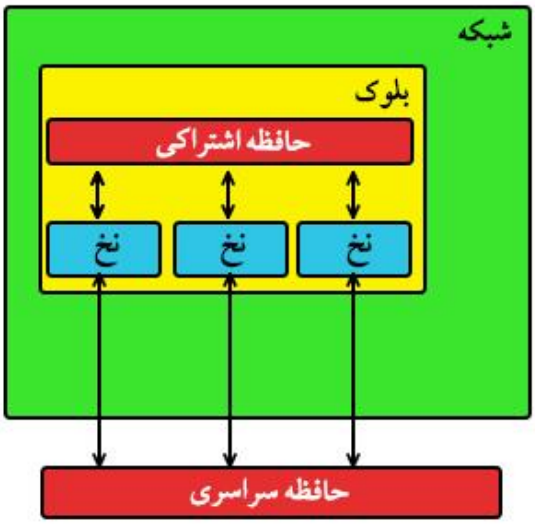
پردازنده گرافیکی	ظرفیت محاسباتی
Tesla K40	3.5
Tesla K20	3.5
Tesla K10	3
Tesla C2070	2
Tesla C1060	1.3

۲-۳-۶- چارچوب کودا

چارچوب کودا[1] یک چارچوب نرم‌افزاری برای دسترسی به واحدهای پردازنده گرافیکی می‌باشد. سابقاً برای دسترسی به هسته‌های پردازشی نیاز بود که محاسبات عددی در قالب پیکسل‌های از تصاویر وارد پردازنده گرافیکی شده و محاسبات به شکل پردازش تصویر روی مجموعه داده‌های عددی صورت پذیرد. چارچوب نرم‌افزاری کودا یک کتابخانه از مجموعه دستورات جهت انجام پردازش‌های موازی به کمک پردازنده گرافیکی می‌باشد. کودا یک زبان برنامه‌نویسی نیست بلکه یک لایه نرم‌افراری است که سختی و پیچیدگی پردازش موازی در واحد پردازش کارت گرافیک[2] را کاهش می‌دهد. برنامه‌نویسی کودا را می‌توان در کامپایلرهای مختلفی نظیر سی‌پلاس پلاس[3]، سی‌شارپ[4]، جاوا[5] و غیره انجام داد[۲۶و۲۵].

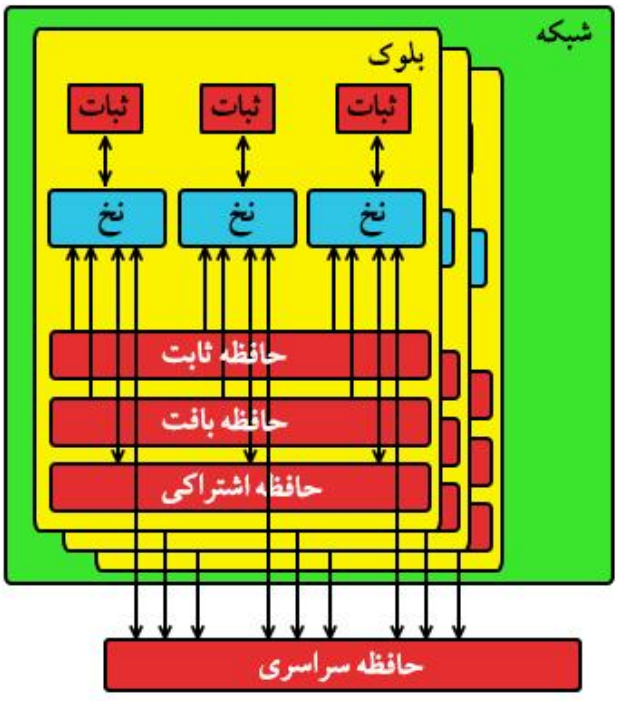
در چارچوب کودا معماری حافظه به شکل سلسله مراتبی است [6] که در شکل(۲-۲۰) یک معماری ساده حافظه نشان داده شده است. در این معماری حافظه، دو حافظه سراسری[7] و اشتراکی[8] نقش مهمی در ذخیره اطلاعات دارند[۱۰]:

[1] CUDA framework
[2] GPU
[3] C++
[4] C#.net
[5] java
[6] memory hierarchy
[7] global memory
[8] share memory



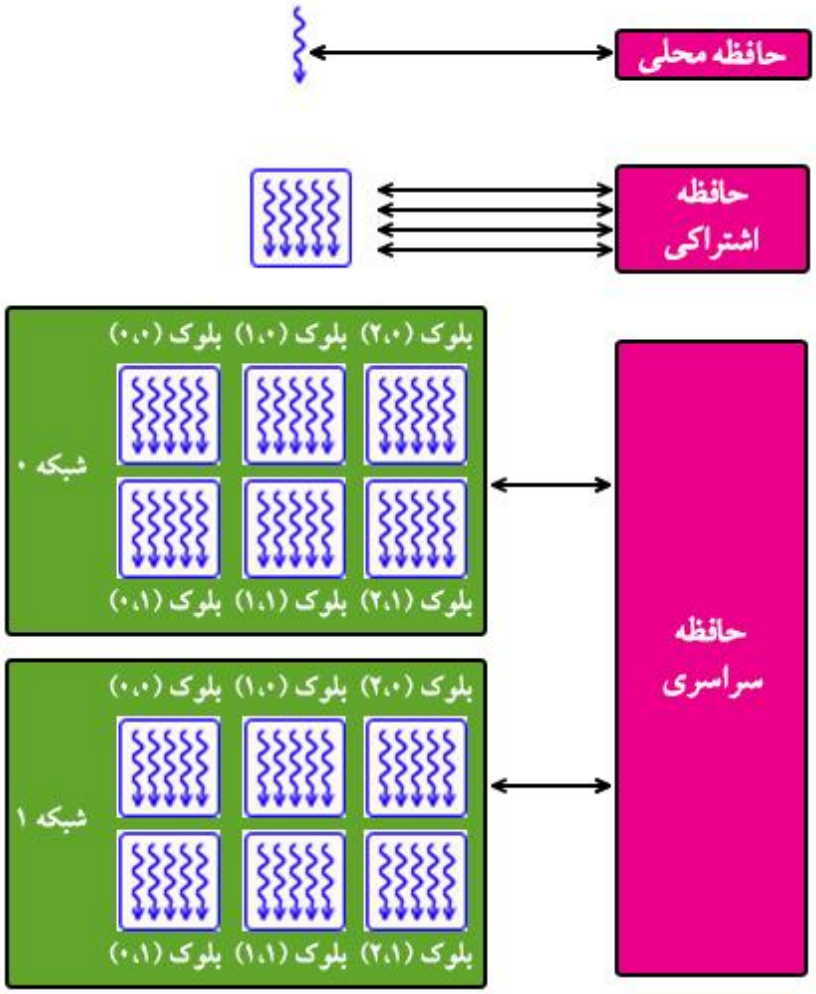
شکل ۲-۱۷: معماری ساده حافظه در چارچوب کودا

در شکل(۲-۲۱) معماری سلسله مراتبی حافظه کودا به شکل کامل‌تری به تصویر کشیده شده است. در این شکل هر پروسه می‌تواند با استفاده از حافظه اشتراکی داده‌ها و اطلاعات خود را با سایر پروسه-های بکار رفته در پردازنده گرافیکی به اشتراک بگذارد و جهت انتقال داده‌ها به پردازنده اصلی در ابتدا داده‌های خود را درون حافظه سراسری کپی نموده و در ادامه اطلاعات از حافظه سراسری به حافظه پردازنده اصلی کپی شوند[۲۷]:



شکل ۲-۱۸: معماری سلسله مراتبی حافظه در چارچوب کودا

در معماری سلسله مراتبی حافظه کودا یک نخ که به مدیریت قسمتی از داده‌های مورد پردازش موازی می‌پردازد در طی روند پروسه اجرای خود می‌تواند از یک حافظه اشتراکی مشترک با سایر نخ‌ها استفاده نماید و در قالبی مانند گرید یا بلاک به تبادل داده‌ها با حافظه سراسری بپردازد. در شکل(۲-۲۳) یک نخ که در حال اجرای یک کرنل است می‌تواند در حین مراحل موازی‌سازی به تبادل داده‌ها با حافظه اشتراکی و سراسری بپردازد[۲۴]:



شکل ۲-۱۹: معماری سلسله مراتبی حافظه در چارچوب کودا و اجرای سلسله مراتبی نخ‌ها

۲-۳-۷- گرید، بلاک، نخ و کرنل

داده ها در پردازنده گرافیکی در قالبهای مختلفی مدیریت می‌شوند. یک نخ[1] کوچکترین جزء برای موازی سازی محسوب می‌شود. نخ‌ها در واقع برای مدیریت زمانی پروسه‌ها بکار گرفته می‌شود.

1 Thread

مجموعه‌ای از نخ‌ها را می‌توان در قالب بلاک‌ها[1] سازماندهی می‌شوند. گرید[2] در پردازنده گرافیکی شامل تعدادی از بلاک‌های پردازنده گرافیکی می‌باشد. بطور کلی می‌توان مجموعه‌ای از نخ‌ها را یک بلاک و مجموعه‌ای از چند بلاک را یک گرید نامید. بلاک ها معمولا زمانبندی نمی‌شوند اگر نخهای که در یک بلاک قرار دارند بیش از اندازه منتظر دریافت داده‌ها از حافظه شوند کودا می‌تواند از زمانبندی بلاک ها استفاده کند. این عملیات منجر می‌شود که کودا بیهوده منتظر دریافت اطلاعات در یک بلاک نشود و سرعت پردازش در واحد پردازش کارت گرافیک کاهش نیابد. اگر کودا از زمانبندی بلاک ها استفاده نماید در زمان سوئیچ بین بلاک ها حافظه‌های اشتراکی تعویض نمی‌شوند[۱۰].

کرنل[3] در واقع قسمت موازی کد برنامه کودا است که توسط مجموعه‌ای از نخ‌ها به اجراء گذاشته می‌شود. در واقع به توابع و برنامه هایی که در واحد پردازش کارت گرافیک قرار دارند و توسط پردازنده اصلی فراخوانی شده و در پردازنده کارت گرافیک اجرا می شوند، کرنل گفته می‌شود.

در زمان اجرای یک کرنل در واحد پردازش کارت گرافیک یک نسخه و کپی از کد برنامه در اختیار تمام نخهای که می‌خواهند این کرنل را اجرا کنند قرار خواهد گرفت ولی هر نخ فقط به داده مختص به خود دسترسی دارد فرآیند ارتباطی بین این نخ ها در واقع با استفاده از حافظه اشتراکی و هماهنگی بین آنها اتفاق می‌افتد. در برنامه نویسی کودا هر نخ شماره منحصر به فردی می‌گیرد که به طور پویا تعیین می‌شود[۲۴].

۲-۴- جمع بندی و نتیجه گیری

این فصل از پایان‌نامه در سه بخش اصلی گردآوری شده است که هدف این سه بخش آشنایی خواننده با مفاهیم الگوریتم تکاملی گرده‌افشانی گل، معماری موازی کودا و کتابخانه کودافای می‌باشد. نتایج بررسی‌ها نشان می‌دهد که الگوریتم گرده‌افشانی گل با تقلید از رفتار گیاهان در گرده‌افشانی می‌تواند مسائل بهینه‌سازی مختلف را به خوبی الگوریتم ذرات و ژنتیک و حتی بهتر از آنها حل نماید علاوه بر آن بررسی معماری موازی‌سازی پردازنده‌گرافیکی نشان می‌دهد که چارچوب موازی و هسته‌های متعدد آن می‌تواند بستری مناسب برای موازی‌سازی الگوریتم‌های مختلف از جمله الگوریتم گرده-افشانی گلها باشد. در این فصل هر کدام از موارد گفته شده تا حدودی تشریح شد و در فصل پیش‌رو تلاش خواهیم نمود که مروری مختصر بر پژوهش‌های مرتبط با موازی داشته باشیم.

1 Block
2 Grid
3 Kernel

فصل ۳- مروری بر مطالعات مرتبط

۳-۱- مقدمه

معماری موازی‌سازی کودا یکی از چارچوبهای مناسب بر تسریع و شتاب دادن الگوریتم‌های است که ماهیت موازی دارند. الگوریتم‌های تکاملی یکی از روش‌های حل مسئله است که جمعیت اولیه آنها می‌توانند مستقلاً فضای جستجوی مسئله را مورد پیمایش قرار دهند و از این رو ساختاری مناسب جهت موازی‌سازی به کمک معماری کودا دارند. در این فصل به جای پرداختن به موضوعات فروان که باعث می‌شود جزئیات کار پنهان شود تمرکز خود را بر دو الگوریتم موازی شده ماهی مصنوعی و زنبور عسل گذاشته‌ایم که مطالعه آنها می‌توان درک بهتری از موازی‌سازی این الگوریتم‌ها فراهم آورد.

۳-۲- الگوریتم موازی هوش دسته جمعی ماهی مصنوعی

الگوریتم هوش دسته جمعی ماهی مصنوعی[1]، توسط لی در سال ۲۰۰۲ توسعه یافته است. این الگوریتم دسته جمعی، یک الگوریتم بهینه جستجوی تصادفی مبتنی بر شبیه سازی ازدحام ماهی‌ها می‌باشد. در الگوریتم هوش دسته جمعی ماهی، هر ماهی مصنوعی، ارزش بهینه محلی مکان خود را به وسیله طعمه جستجو می‌کند و با رفتاری مبتنی بر دنباله روی از سایر ماهی ها و پس از چند بار تکرار این مکانیزم، تمام ماهی‌ها و دسته‌های آنها به مقدار بهینه سراسری مطلوب دستیابی پیدا می-کنند.[۳۰].

۳-۲-۱- هوش دسته جمعی ماهی در معماری پردازنده گرافیکی

در این بخش، یک پیاده سازی از الگوریتم هوش دسته جمعی ماهی مصنوعی بر روی واحد پردازش گرافیک مورد بررسی قرا می‌گیرد. در واقع در این قسمت به بررسی شتاب دادن به الگوریتم هوش دسته جمعی ماهی مصنوعی با استفاده از امکانات موازی پردازنده گرافیکی می‌پردازیم. در ادامه نشان می‌دهیم اجرای الگوریتم هوش دسته جمعی ماهی مصنوعی پیشنهادی دارای سرعتی به مراتب بیشتر از الگوریتم هوش دسته جمعی ماهی مصنوعی استاندارد است.

۳-۲-۱-۱- تعریف متغییرها در الگوریتم هوش دسته جمعی ماهی مصنوعی

فرض کنید تابع شایستگی را در مسئله به شکل F(X)، دامنه متغییر X در بازه [r , r–]، اندازه ابعاد مسئله D و جمعیت اولیه مسئله N فرض شود، طبق تعاریف الگوریتم هوش دسته جمعی ماهی مصنوعی متغیر های زیر تعریف می‌شود:

- X : آرایه‌ی که وضعیت ماهی ها را نشان می‌دهد.
- XP : آرایه‌ی که وضعیت طعمه ها و غذا ها را نشان می‌دهد.
- XS : آرایه‌ی که وضعیت دسته جمعی ماهی ها را نشان می‌دهد.
- XF : آرایه‌ی که دنباله روی ماهی ها را نشان می‌دهد.
- DIST : آرایه‌ی که فاصله بین ماهی‌ها را نشان می‌دهد .
- TESTF : آرایه‌ی که شایستگی ماهی‌ها را نشان می‌دهد.

اندازه‌ی آرایه‌های XF,XS,XP,X همگی N*D، اندازه آرایه‌ی DIST برابر N*N ، اندازه آرایه‌ی TESTOF برابر با N*I است. آرایه‌های X,XF,XS,XP می‌توانند همانند یک ماتریس با سطرهای N و ستون های D نمایش داده شوند. تعریف اندازه بلوک و گریدها یک عامل اصلی و تعیین کننده جهت

1 Artificial Fish Swarm Algorithm

اجرای کرنل به شکل موازی است. تعاریفی که برای موازی سازی ارایه شده است در روابط(۳-۱)، (۳-۲) و(۳-۳) عنوان شده است[۳۴]:

$block_1 = (BS, BS, 1)$ **رابطه ۳-۱**

رابطه ۳-۲

$$grid_1 = \left(\frac{WIDTH}{BS} + 1\,, \frac{HEIGHT}{BS} + 1\,, 1\right)$$

رابطه ۳-۳

$$grid_2 = \left(\frac{N}{BS} + 1\,, \frac{N}{BS} + 1\,, 1\right)$$

در روابط بالا BS عرض و ارتفاع یک بلاک، WIDTH و HEIGHT تعداد نخ‌های هر بلاک در راستای افقی و عمودی هر بلاک است از نقطه نظر کلی این الگوریتم به دنبال Ymin است که یک بهینه حداقلی سراسری است، همچنین حداکثر تعداد تکرار در الگوریتم را نیز با iterNum مشخص می‌شود.

۳-۲-۱-۲- تولید اعداد تصادفی در الگوریتم هوش دسته جمعی ماهی مصنوعی

در این الگوریتم اعداد تصادفی توسط CPU تولید شده و به GPU منتقل می شود.

۳-۲-۲- چارچوب الگوریتم موازی هوش دسته جمعی ماهی [۱]

در شکل(۳-۱)، چارچوب[۲] و شبه کد الگوریتم موازی هوش دسته جمعی ماهی به طور کلی نشان داده شده است همانگونه که در شکل مذکور مشخص است این شبه کد دارای کرنل های مختلفی است که زیر حلقه‌های اصلی به اجراء گذاشته می‌شوند[۳۴].

[1] GPU-AFSA
[2] Framework

Algorithm 1 Framework of GPU-AFSA

Initialize the states of AFs randomly
Initialize the random number array
Initialize the parameter of AFSA
Transfer data from CPU to GPU
$Y_{min} \leftarrow +\infty$
for $iter \leftarrow 1$ **to** $iterNum$
    do Calculate fitness values of all AFs: TESTF
        Update the optimization result: Y_{min}
        Calculate distance between AFs: DIST
        Execute fish preying process
        Execute fish swarming process
        Execute fish following process
        Update the state of each AF: X
Transfer Y_{min} back to CPU and output

شکل ۳-۱: شبه کد الگوریتم موازی هوش دسته جمعی ماهی [۳۴]

۳-۲-۳- کرنل های الگوریتم موازی هوش دسته جمعی ماهی

در این بخش کرنل‌های بکار رفته در الگوریتم موازی هوش دسته جمعی ماهی نشان داده شده است.

۳-۲-۳-۱- کرنل محاسبه شایستگی ماهی ها

در الگوریتم موازی هوش دسته جمعی ماهی، نیاز است مقادیر بلاک ها و گریدها به شکل مناسبی ارایه شود .در روش ما وقتی یک کرنل با N نخ به اجراء در آید باید روشی وجود داشته باشد تا هر ماهی مقدار شایستگی خود را محاسبه نماید این کرنل در شکل(۳-۲) نشان داده شده است[۳۴].

Algorithm 2 Calculate fitness values of all AFs

Determine which AF's fitness value to calculate: i
Apply arithmetical operations to $X[i]$ and store the calculated fitness value in $TESTF[i]$

شکل ۳-۲: کرنلی که شایستگی ماهی ها را محاسبه می‌کند. [۳۴]

۳-۲-۳-۲- کرنل محاسبه بهترین مقدار بهینه

یکی از کرنل‌های که در الگوریتم موازی هوش دسته جمعی ماهی بکار گرفته شده است، محاسبه بهترین مقدار بهینه سراسری است که تاکنون حاصل شده است. برای این منظور یک کرنل ساده نیاز است تا آرایه‌ی TESTF[i] را مورد پیمایش قرار دهد و بهترین شایستگی را در متغییر Y_{min} دهند.

۳-۲-۳-۳- کرنل محاسبه فاصله بین ماهی ها

یکی از کرنل‌های که در الگوریتم موازی ماهی ارایه شده است محاسبه فاصله بین ماهی‌ها در هوش دسته جمعی ماهیان است. در این کرنل هر نخ فاصله بین دو ماهی مختلف را محاسبه می‌کند. این

کرنل در شکل(۳-۳) نشان داده شده است. البته می‌توان در این کرنل هر نخ را به یک ماهی منتسب کرد تا فاصله بین آنها محاسبه شود.

Algorithm 3 Calculate distance between AFs

Determine which AF pair's distance to calculate: (i_1, i_2)
if $i_1 \geq N$ **or** $i_2 \geq N$ **or** $i_2 < i_1$ **then return**
Calculate the distance of (i_1, i_2)
Store the result to $DIST[i_1][i_2]$ and $DIST[i_2][i_1]$

شکل ۳-۳: کرنلی که فاصله بین ماهی ها را محاسبه می‌کند. [۳۴]

۳-۲-۳-۴- کرنل دنبال کردن طعمه

هر ماهی برای یافتن غذا نیاز به 2*tryNum+1 عدد تصادفی جهت یافتن غذا دارد در این روش tryNum تعداد تلاش ماکزیمم جهت یافت غذا است. در شکل(۳-۴) این کرنل و شبه کد آن نشان داده شده است. در این تکنیک ما یک ماکرو در زبان سی تعریف کرده‌ایم که به صورت PREY(pr1 , pr2 , A , I) است که در آن هر ماهی با وضعیت X[i] در جهت یافتن غذا تلاش می‌کند و دو شاخص تصادفی به نام pr1 و pr2 برای کسب اعداد تصادفی بار گذاری شده در واحد پردازش کارت گرافیک مورد استفاده قرار می‌گیرد. در این روش اگر یک تلاش جهت یافتن غذا به نتیجه برسد وضعیت غذایابی یا صید ماهی در A[i] ذخیره می‌شود[۳۴].

Algorithm 4 Fish preying process

Determine which AF's prey process to execute: i
$preySuccess \leftarrow false$
for $t \leftarrow 1$ **to** $tryNum$
  do $PREY(pr_{2t-1}, pr_{2t}, XP, i)$
if $preySuccess = false$
  then $XP[i] \leftarrow X[i] + RAND[r+i] * Step$

شکل ۳-۴: کرنل دنبال کردن طعمه توسط ماهی ها [۳۴]

۳-۲-۳-۵- کرنل هوش دسته جمعی ماهی

در این کرنل ساختار هوش دسته جمعی و فرآیند حرکت دسته جمعی ماهیان انجام می‌شود این ساختار در کرنل شکل(۳-۵) نشان داده شده است.

Algorithm 5 Fish swarming process

Determine which AF's prey process to execute: i
$tag \leftarrow false$
if $i \geq N$
  then return
Calculate n_f and X_c
if satisfies swarming condition
  then $tag \leftarrow true$
    $XS[i] \leftarrow X[i] + RAND[rs + i] * Step * \frac{X_c - X[i]}{\|X_c - X[i]\|}$
if $tag = false$
  then Execute preying process.

شکل ۳-۵: کرنل حرکت دسته جمعی ماهیان [۳۴]

۳-۲-۳-۶- کرنل دنباله روی ماهیان

در این کرنل ماهیان به سمت کمترین مقدار که در اینجا بهترین شایستگی را دارند حرکت می‌کنند. این کرنل در شکل(۳-۶) نشان داده شده است[۳۴].

Algorithm 6 Fish following process

Determine which AF's prey process to execute
$tag \leftarrow false$
if $i \geq N$
  then return
Find the AF's companion with minimal fitness
Count the number of the AF's companions n_f
if $nf \neq 0$ **and** $F_{min} < TESTF[i]$ **and** $n_f/N < \delta$
  then $tag \leftarrow true$
    $XS[i] \leftarrow X[i] + RAND[rf + i] * Step * \frac{X_{min} - X_i}{\|X_{min} - X_i\|}$
if $tag = false$ **then** Execute preying process.

شکل ۳-۶: شبه کد و کرنل حرکت دنباله دار و دسته جمعی ماهیان [۳۴]

۳-۲-۳-۷- کرنل به روز رسانی

در این کرنل وضعیت تک تک ماهیان در هر تکرار به روزرسانی می‌شود. این کرنل در شکل(۳-۷) نشان داده شده است.

Algorithm 7 Update the state of each AF

Determine which AF's position to update: i
Calculate the fitness value of $XP[i]$ and
    store the result in shared memory $pF[i]$
Calculate the fitness value of $XS[i]$ and
    store the result in shared memory $sF[i]$
Calculate the fitness value of $XF[i]$ and
    store the result in shared memory $fF[i]$
Store the best state of the above three to $X[i]$

شکل ۳-۷: کرنل به روز رسانی وضعیت ماهیان [۳۴]

۳-۲-۴- معیارها و توابع ارزیابی در الگوریتم موازی هوش دسته جمعی ماهی

یکی از روش های ارزیابی الگوریتم های هوش دسته جمعی استفاده از توابع ارزیابی است در این قسمت از توابع بنجمارک Sphere، Rastrigin، Griewangk و Rosenbrock استفاده شده است. در آزمایشات و شبیه سازی ها D=50، iterNum=100 و δ بین ۳.۵ تا ۰.۶۱۸ انتخاب شده است. مقدار متوسط زمان اجراء و مقادیر بهینه متوسط در جدول(۳-۱)، (۳-۲)، (۳-۳) و(۳-۴) نشان داده شده است[۳۴].

جدول ۳-۱: نتایج اجرای الگوریتم موازی و استاندارد ماهی با تابع ارزیابی Sphere با D=50

N	CPU times	GPU times	Speed-up	CPU result	GPU result
1500	140.042	5.810	24.103	1.28e-005	1.16e-005
2000	254.140	8.754	29.031	4.23e-006	3.37e-006
2500	358.110	12.778	28.025	1.95e-006	1.77e-006
3000	561.158	17.431	32.192	2.27e-007	2.39e-007

جدول ۳-۲:نتایج اجرای الگوریتم موازی و استاندارد ماهی با تابع ارزیابی Rastrigin با D=50

N	CPU times	GPU times	Speed-up	CPU result	GPU result
1500	147.859	5.881	25.140	1.69e-001	1.80e-001
2000	237.621	8.822	26.933	1.22e-001	1.12e-001
2500	382.117	12.854	29.728	6.35e-002	5.85e-002
3000	515.869	17.498	29.481	2.45e-002	2.45e-002

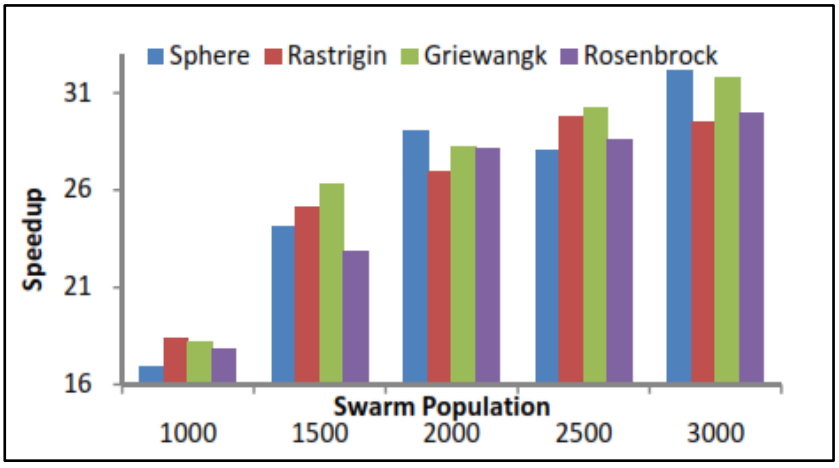
جدول ۳-۳:نتایج اجرای الگوریتم موازی و استاندارد ماهی با تابع ارزیابی Griewangk با D=50

N	CPU times	GPU times	Speed-up	CPU result	GPU result
1500	۱۵۴.۷۹۹	5.886	26.301	1.79e-007	1.19e-007
2000	249.339	8.823	28.260	2.55e-010	2.59e-010
2500	387.780	12.831	30.218	2.61e-010	0
3000	555.247	17.460	31.801	0	0

جدول ۳-۴:نتایج اجرای الگوریتم موازی و استاندارد ماهی با تابع ارزیابی Rosenbrock با D=50

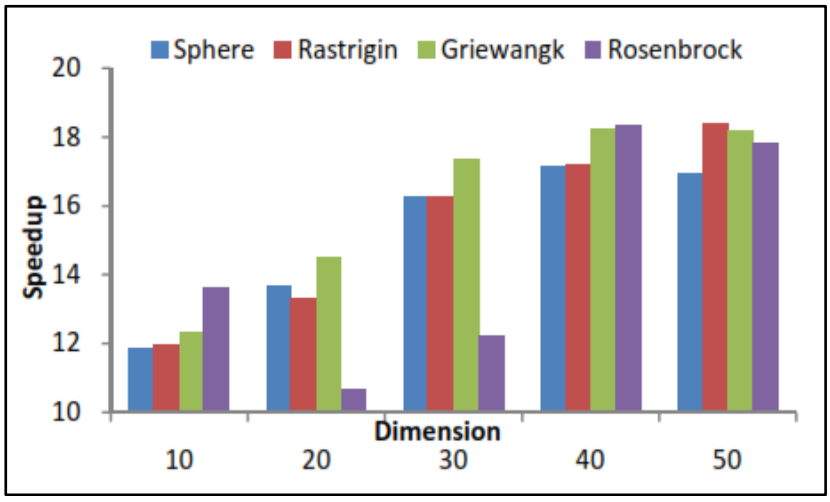
N	CPU times	GPU times	Speed-up	CPU result	GPU result
1500	134.151	5.872	22.844	1.63e-003	1.78e-003
2000	248.331	8.811	28.183	1.31e-003	1.28e-003
2500	367.370	12.834	28.623	2.05e-004	1.98e-004
3000	523.468	17.487	29.934	2.27e-005	3.71e-005

در شکل‌های(۳-۸) و (۳-۹) چهار تابع ارزیابی ریاضی، معیارهای جمعیت اولیه و تعداد ابعاد مسئله برای ارزیابی الگوریتم موازی هوش دسته جمعی ماهی و الگوریتم استاندارد هوش دسته جمعی ماهی مورد استفاده قرار گرفته است. در شکل(۳-۸) میزان شتاب بر حسب جمعیت اولیه نشان داده شده است با توجه به این شکل با افزایش جمعیت اولیه میزان شتاب الگوریتم موازی نیز بیشتر شده است[۳۴].



شکل ۳-۸ : رابطه شتاب در الگوریتم موازی ماهی و تعداد جمعیت اولیه [۳۴]

در شکل(۳-۹) رابطه شتاب الگوریتم موازی و تعداد ابعاد مسئله مورد مطالعه قرار گرفته است به طور کلی با افزایش ابعاد مسئله شتاب الگوریتم موازی نیز افزایش یافته است[۳۴].



شکل ۳-۹ : رابطه شتاب در الگوریتم موازی ماهی و تعداد ابعاد مسئله [۳۴]

برای بررسی میزان تاثیر افزایش ابعاد بر اندازه شتاب الگوریتم موازی میزان جمعیت اولیه را در چهار تابع ارزیابی ثابت فرض می‌کنیم و بر تعداد ابعاد مسئله می‌افزایم و تاثیر آن را روی شتاب الگوریتم موازی بررسی می‌کنیم در جداول (۳-۵)، (۳-۶)، (۳-۷)، (۳-۸) و (۳-۹) این مقایسه ها نشان داده شده است و در آنها تعداد جمعیت اولیه ثابت و برابر N=1000 در نظر گرفته شده است.

جدول ۳-۵:نتایج اجرای الگوریتم موازی و استاندارد ماهی با تابع ارزیابی Sphere با جمعیت اولیه N=1000

D	CPU times	GPU times	Speed-up	CPU result	GPU result
20	25.282	1.852	13.650	1.00e-003	3.04e-003
30	40.018	2.463	16.248	1.40e-005	4.95e-005
40	51.108	2.978	17.161	9.25e-006	9.97e-006
50	59.856	3.536	16.927	2.67e-004	2.28e-004

جدول ۳-۶:نتایج اجرای الگوریتم موازی و استاندارد ماهی با تابع ارزیابی Rastrigin با جمعیت اولیه N=1000

D	CPU times	GPU times	Speed-up	CPU result	GPU result
20	24.978	1.878	13.297	8.93e-003	5.67e-003
30	40.664	2.497	16.281	1.54e-002	4.72e-002
40	52.182	3.030	17.218	2.18e-002	1.82e-002
50	66.542	3.613	18.413	2.85e-001	3.14e-001

جدول ۳-۷:نتایج اجرای الگوریتم موازی و استاندارد ماهی با تابع ارزیابی Griewangk با جمعیت اولیه N=1000

D	CPU times	GPU times	Speed-up	CPU result	GPU result
20	27.405	1.892	14.480	3.26e-004	3.65e-004
30	43.653	2.514	17.361	1.78e-007	1.54e-007
40	55.574	3.046	18.242	2.38e-006	0
50	66.044	3.630	18.188	6.20e-006	1.79e-006

جدول ۳-۸:نتایج اجرای الگوریتم موازی و استاندارد ماهی با تابع ارزیابی Rosenbrock با جمعیت اولیه N=1000

D	CPU times	GPU times	Speed-up	CPU result	GPU result
20	27.405	1.892	14.480	3.26e-004	3.65e-004
30	43.653	2.514	17.361	1.78e-007	1.54e-007
40	55.574	3.046	18.242	2.38e-006	0
50	66.044	3.630	18.188	6.20e-006	1.79e-006

۳-۲-۵- نتایج

از نتایج آزمایشات و پیاده سازی ها نتیجه می‌شود که الگوریتم موازی هوش دسته جمعی ماهی مانند الگوریتم استاندارد هوش دسته جمعی ماهی نتایج مطلوبی را ارایه می‌دهد و عملکرد الگوریتم موازی هوش دسته جمعی ماهی در کل قابل قبول است. نتایج نشان داد که الگوریتم موازی هوش دسته

جمعی ماهی بسیار سریعتر از الگوریتم استاندارد هوش دسته جمعی ماهی است. در آزمایش اول، بهترین عملکرد محاسباتی دارای شتابی به میزان ۳۲ برابر است. در آزمایش دوم، بهترین عملکرد محاسباتی با تابع Rastrigin حاصل شد که شتاب الگوریتم موازی در این تابع حدود ۱۸.۴ شد. در کل میزان شتاب دهی الگوریتم موازی هوش دسته جمعی ماهی در آزمایش دوم که بر حسب تغییر ابعاد بود از آزمایش اول که بر حسب جمعیت اولیه بود کمتر است. با توجه به میزان شتاب و اندازه ابعاد نسبت به جمعیت اولیه نتیجه می‌شود که محاسبات سریالی با افزایش ابعاد نیز به همان نسبت افزایش می‌یابد که عاملی مهم در کاهش شتاب محاسبات است.

در این پژوهش استفاده از کرنل‌های مختلف به علت سویچ‌ها و انتقال‌های بین حافظه و پردازنده گرافیکی بر میزان شتاب تاثیر منفی می‌گذارد و استفاده از یک کرنل واحد برای پیاده‌سازی موازی می‌تواند تاثیر بیشتری در سرعت و کاهش زمان اجراء داشته باشد.[۳۴].

۳-۳- الگوریتم موازی هوش دسته جمعی زنبور عسل

الگوریتم زنبورعسل[1] روشی مبتنی بر جمعیت و یک الگوریتم محاسباتی است که از رفتار طبیعی زنبور عسل برای یافتن راه‌حل نزدیک به بهینه در مسائل جستجو و بهینه سازی الهام می‌گیرد[۳۵]. در این بخش الگوریتم زنبور عسل مبتنی بر چارچوب کودا[2] توسعه داده می‌شود. در این بخش عملکرد الگوریتم زنبور عسل مبتنی بر چارچوب کودا را با انجام برخی آزمایشات بر اساس مسائل بهینه‌سازی متعدد و مشهور مورد ارزیابی قرار گرفته می‌شود. نتایج نشان می‌دهند که در تعداد زیادی از مسائل بهینه‌سازی مختلف، الگوریتم زنبور عسل مبتنی بر چارچوب کودا به طور قابل توجهی عملکردی بهتر از الگوریتم زنبور عسل استاندارد دارد.

در این الگوریتم نخ‌ها در یک بلوک به چندین کلونی دسته‌بندی می‌شوند و هر نخ به یک زنبورعسل برای جستجوی راه‌حل اختصاص داده می‌شود. الگوریتم پیشنهادی یک بلوک را به کلونی‌های مختلف با شناسه نخ تقسیم می‌کند، و به طور مستقل الگوریتم زنبورها را اجرا می‌کند.[۳۵].

۳-۳-۱- بهینه‌سازی کلونی زنبورعسل

به طور کلی، کلونی زنبور عسل شامل سه نوع زنبور بالغ کارگران، زنبورهای نر، و یک ملکه می‌باشد. اگر چه هر عضو در کلونی زنبور عسل یک وظیفه مشخص برای انجام دارد، بسیاری از زنبورهای کارگر برای تکمیل وظایف پیچیده نیاز به همکاری دارند، مانند ساختن لانه، یافتن و جمع‌آوری غذا و پرورش نوزادان. علاوه بر این، زنبورهای فردی (کارگران، زنبورهای نر، و ملکه‌ها) نمی‌توانند بدون حمایت کلونی زنده بمانند. بنابراین، زنده ماندن و تولید مثل نیاز به ترکیب تلاش‌های کل کلونی دارد. برای کاوش، کلونی زنبور عسل به صورت انتخابی منابع شهد در دسترس را کاوش می‌کنند. این فرایند

[1] Bees Algorithms
[2] CUBA

با زنبورهای دیده‌بان آغاز می‌شود که برای جستجوی دسته‌های گل امیدبخش فرستاده شده‌اند. زنبورهای دیده‌بان به طور تصادفی از یک دسته به دیگری حرکت می‌کنند. زنبورهای دیده‌بان بعد از برگشت به کندو، اگر دسته‌ای پیدا کنند که در آستانه کیفیت خاص ارزیابی شود، آن‌ها شهد یا گرده خود را واگذار می‌کنند و به "طبقه رقص" برای انجام رقص شناخته شده به عنوان " رقص چرخشی " می‌روند.

هر زنبور عسل زنبوری (زنبور هم لانه) را که در حال حاضر یک دسته از گل‌ها را کشف کرده است، دنبال خواهد کرد. به محض ورود، زنبور جستجو کننده غذا باری از شهد را می‌گیرد و به کندو برای گذاشتن شهد به کنار زنبور عسل ذخیره کننده مواد غذایی برمی‌گردد. پس از این که او مواد غذایی را گذاشت، زنبور می‌تواند (الف) منبع غذایی را رها کند و دوباره دنبال کننده غیرمتعهد شود، (ب) ادامه به کاوش منبع غذایی بدون بکارگیری هم لانه، یا (ج) رقص و در نتیجه جذب هم لانه قبل از بازگشت به منبع غذایی. زنبورعسل یکی از گزینه‌های بالا را با یک احتمال معین انتخاب می‌کند. در منطقه رقص، زنبور عسل رقصنده مناطق غذای مختلف را " تبلیغ " می‌کند. مکانیسم‌هایی که زنبور تصمیم می‌گیرد یک رقصنده خاص را دنبال کند، به خوبی شناخته نشده‌اند، اما آندر نظر گرفته شده است که " بکارگیری زنبور عسل‌ها همیشه تابعی از کیفیت منبع غذایی است ". همچنین اشاره شده است که همه زنبورها جستجوی غذا را به طور همزمان شروع نمی‌کنند.[۳۵].

۳-۳-۲- الگوریتم کلونی زنبور عسل

الگوریتم کلونی زنبورها روش مبتنی بر جمعیت است که برای یافتن یک راه‌حل نزدیک به بهینه در مسائل جستجو بکار گرفته می‌شود. این روش از رفتار زنبور عسل در طبیعت الهام گرفته است. در این الگوریتم نیازاست چندین پارامتر الگوریتم از جمله ، n (تعداد زنبورهای دیده‌بانی)، m (تعدادی مکان‌های انتخاب شده از n مکان بازدید شده)، e (تعداد بهترین مکان از m مکان انتخاب شده)، nep (تعداد زنبورهای بکارگیری شده برای مکان‌های e)، nsp (تعداد زنبورهای بکارگیری شده برای (m-e) مکان انتخاب شده)، ngh (همسایگی این مکان‌ها برای اعزام نقاط جدید) و معیار توقف تعیین شود. در ابتدا الگوریتم با n زنبور عسل دیده‌بانی که به طور تصادفی در دامنه جستجوی مسئله قرار دارند ، آغاز می‌شوند. فلوچارت الگوریتم زنبور عسل در شکل زیر نشان داده شده است.

مراحل کلی الگوریتم زنبور به شرح زیر است :

- مقداردهی اولیه جمعیت با راه‌حل‌های تصادفی.
- ارزیابی برازندگی جمعیت.
- تا زمانی که معیار توقف ارضا نشود

۱- انتخاب مکان‌هایی برای جستجوی همسایگی.

۲- بکار گیری زنبورها برای مکان‌های انتخاب شده (زنبورهای بیشتر برای بهترین مکان‌های e) و ارزیابی برازندگی.
۳- انتخاب شایسته ترین زنبور عسل از هر کلونی.
۴- اختصاص زنبورهای باقی مانده به جستجوی تصادفی و ارزیابی برازندگی آن‌ها.

- پایان حلقه

به منظور افزایش دقت جستجو و اجتناب از محاسبات اضافی، دکتر فام یک نسخه اصلاح شده ارائه کرد [۳۷]، که در آن دو روش جدید به شرح زیر است معرفی شده‌اند.

۳-۳-۲-۱- کاهش همسایگی

اندازه $a=\{a_1, \ldots a_n\}$ از دسته‌های گل در ابتدا به یک مقدار بزرگ تنظیم می‌شوند. برای هر متغیر a_i، آن به شکل رابطه ۳-۴ تنظیم شده است.

رابطه ۳-۴
$$a_i(t) = ngh(t)*(max_i - min_i)$$
$$ngh(0)=1.0$$

t نشان دهنده تکرار tام از حلقه اصلی الگوریتم زنبورها می‌باشد. اندازه دسته بدون تغییر نگه داشته می‌شود تا زمانی که فرایند جستجوی محلی منجر به امتیاز بالاتر برازندگی شود.جستجوی محلی در ابتدا بر روی یک همسایگی بزرگ (برابر با طیف وسیعی از جستجوی سراسری) تعریف شده است، و دارای ویژگی اکتشافی زیادی است. فرایند جستجوی محلی، هر مکان بهتر با برازندگی بالاتر را می‌یابد، آن اندازه ngh را بدون تغییر نگه می‌دارد. اگر هیچ بهبودی در طی این مرحله نباشد، اندازه ngh کاهش می‌یابد. فرمول به روز رسانی شده در رابطه(۳-۵) نشان داده شده است[۳۵]:

رابطه ۳-۵
$$ngh(i+1)=0.8*ngh(i) \quad if \ no \ improvement$$
$$ngh(i+1)= ngh(i) \quad else$$

۳-۳-۲-۲- رها کردن مکان

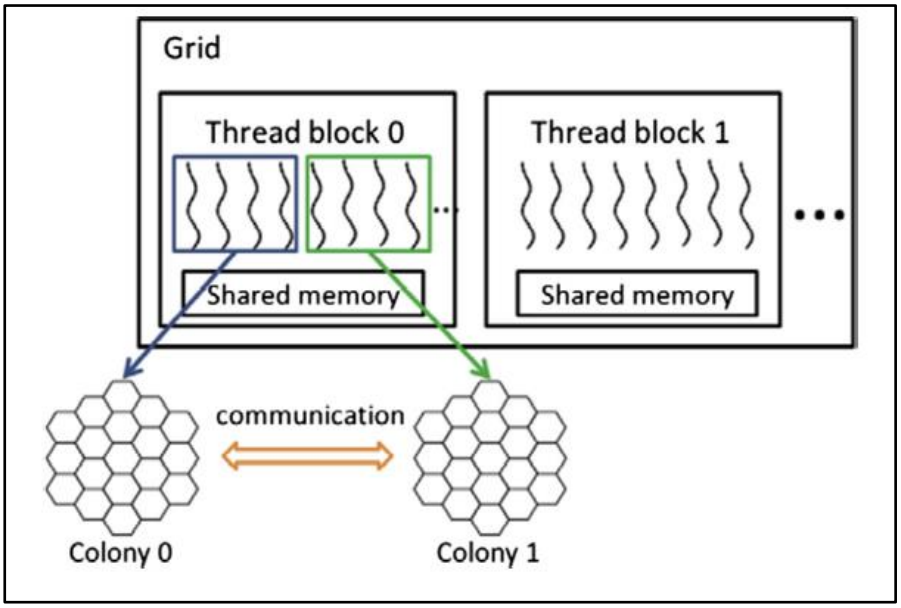
زمانی که بعد از چند بار هیچ بهبود برازندگی (stlim: حد چرخه‌های ایستا برای رها کردن مکان) جستجوی محلی حتی با استفاده از روش کاهش همسایگی حاصل نشود، این به معنی اینست که روش جستجوی محلی شاید به بالای قله برزندگی محلی رسیده است، به عبارت دیگر، هیچ پیشرفتی حاصل نخواهد شد. برای کارایی، اکتشاف دسته متوقف شده است. اگر هیچ برازندگی بهتری از مکان‌های دیگر در طی عمل جستجوی تصادفی باقی مانده تولید نشود، این مکان رها می‌شود.اگر چه محققان بسیاری مدل‌های جدید مبتنی بر زنبور عسل را ارائه نموده‌اند، این مقاله بر اساس مدل پیشنهاد شده توسط D.T.Pham ارائه شده است[۳۷].

۳-۳-۳- الگوریتم موازی زنبور بر روی GPU

کلید اصلی در افزایش شتاب یک الگوریتم، سطح موازی‌سازی آن الگوریتم است. در الگوریتم استاندارد زنبور، بیشتر بارهای محاسباتی در فاز جستجوی همسایگی می‌باشد. یک روش ساده این است که روش جستجوی همسایگی به عنوان یک تابع از نوع کرنل در یک حلقه به اجراء در آید. عدد بهینه اندازه همسایگی با توجه به ویژگی‌های مختلف از توابع، متغیر است. با این حال، اگر اندازه همسایگی بزرگتر از تعداد کل نخ‌ها در پردازنده گرافیکی نباشد، شتاب الگوریتم به چشم نمی‌آید یکی دیگر از راه‌حل‌های معمول " چند کلونی " است که به معنی اینست که باید بسیاری از الگوریتم‌های زنبور به طور مستقل در هر نخ اجرا شوند. دو عیب عمده در این تکنیک وجود دارد. عیب اول این است که هر نخ شامل تعدادی شرایط محدودکننده است. بدیهی است، جلوگیری از شرایط محدودکننده دشوار است، به طوری که سربار در این حالت بسیار هزینه بر خواهد بود. دوم این است که ارتباط میان نخ‌ها پس از یک بار انجام الگوریتم پیچیده‌تر خواهد بود. به دلایل فوق، پیشنهاد می-شود که الگوریتم زنبورهای جدید را از کلونی‌های متعدد موازی با بهره‌وری خوب طراحی شود.

۳-۳-۴- بررسی سیستم موازی الگوریتم زنبورها

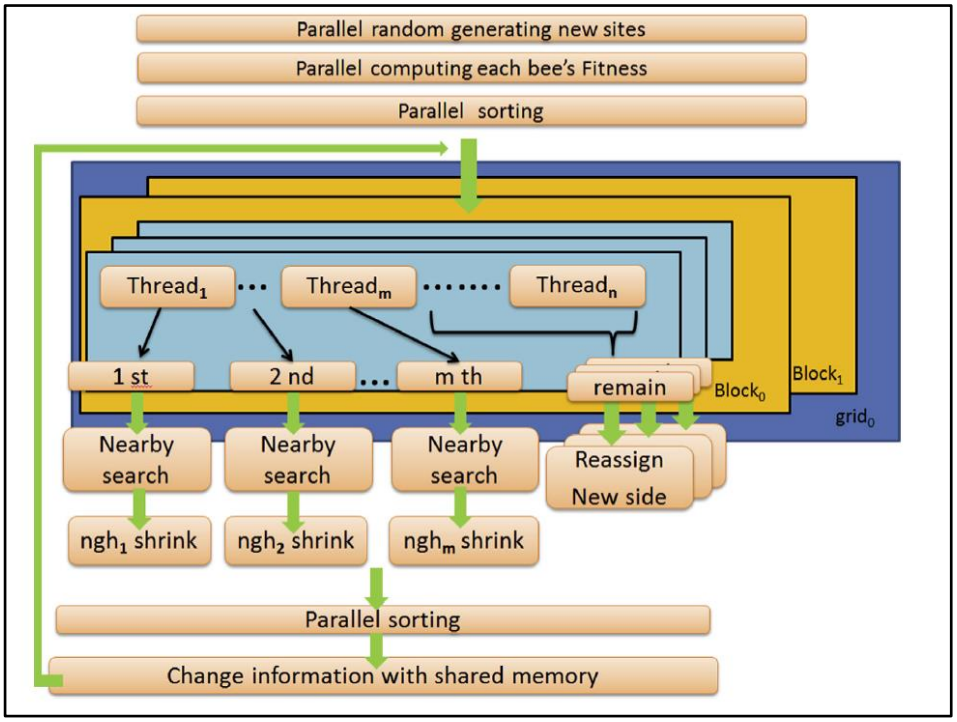
چارچوب برنامه نویسی کودا برای پیاده‌سازی الگوریتم زنبورهای چندکلونی در واحد پردازش را به اختصار کوبا (الگوریتم موازی زنبور در واحد پردازش کارت گرافیک) می‌نامیم. شکل(۳-۱۱) یک مرور کلی از چارچوب کوبا را نشان می‌دهد. در چارچوب کوبا نخ‌ها در یک بلوک به چند کلون طبقه‌بندی شده‌اند. به عبارت دیگر، هر نخ به یک زنبور عسل برای جستجوی راه‌حل برای کلونی خود اختصاص داده است. بلوک به کلونی‌های مختلف اختصاص داده شده است و با شناسه نخ تقسیم بندی شده است و به طور مستقل الگوریتم زنبورها در این واحدها اجراء می‌شود. هنگامی که یک تکرار به پایان رسید، کوبا اطلاعات بین کلونی‌ها در همان بلوک را با استفاده از حافظه مشترک تغییر می‌دهد. از آنجایی که تاخیر ارتباطات بین کلونی‌ها برای زمان همگرایی الگوریتم بسیار مهم است، در اشتراک گذاری اطلاعات بین نخ‌ها از حافظه سراسری که کند تر از حافظه اشتراکی استفاده نمی‌شود. به عبارت دیگر، اگر کلونی‌ها در بلوک‌های مختلف باشند، کلونی‌ها با یکدیگر ارتباط برقرار نخواهند کرد چرا که حافظه مشترک با نخ‌ها در همان بلوک به اشتراک گذاشته شده است (شکل ۳-۱۱)



شکل ۳-۱۰ : چارچوب الگوریتم موازی زنبور در واحد پردازش کارت گرافیک [۳۵]

۳-۳-۵- الگوریتم موازی زنبورها

این بخش الگوریتم موازی زنبوردر در معماری کودا شرح داده می‌شود. برای تغییر الگوریتم استاندارد زنبور به الگوریتم موازی زنبور و اجرای آن را در معماری کودا ، نیاز به برطرف کردن بـر بسـیاری از مسائل پیاده‌سازی است. در این بخش ایده طراحی الگوریتم موازی زنبور مبتنی بر کـودا نشـان داده شده است، که در شکل(۳-۱۲) نشان داده شده است. همچنین مسـائل مربـوط بـه پیـاده سـازی در جزئیات بررسی می‌شوند. سپس گام‌هـای الگـوریتم و مسـائل مربـوط بـه پیـاده‌سـازی توضـیح داده می‌شوند[۳۵].



شکل ۳-۱۱ : الگوریتم موازی زنبور در معماری کودا [۳۵]

۳-۳-۶- مقداردهی اولیه موازی

در رویکرد BA، مقداردهی اولیه جمعیت و ارزیابی برازندگی جمعیت یک به یک به دست خواهد آمد در BA موازی می‌توان این کار را بصورت موازی و همزمان انجام داد. . از این رو CUBA آنها را از طریق رشته‌های موازی در GPU توزیع و محاسبه خواهد کرد. در مرحله اول، CUBA باید پارامترهای موردنیاز را تنظیم کند، حافظه مشترک اولیه را برای ارتباط اختصاص داد، تعداد کلونی و .... را تنظیم نماید. که مشابه با BA استاندارد است ولی به صورت موازی انجام می شود. علاوه بر این، الگوریتم باید به طور تصادفی مکان‌های جدید را تولید کند و برازندگی هر زنبور عسل در موازی را محاسبه می‌نماید[۳۵].

۳-۳-۶-۱- مرتب‌سازی زوج-فرد

مرتب‌سازی برازندگی همه جمعیت‌ها برای دریافت بهترین m مکان لازم است. به دلیل این که اندازه داده‌های مرتب‌سازی در این برنامه نسبت به دیگران کوچک است، کلونی‌ها در همان بلوک با استفاده از الگوریتم مرتب‌سازی زوج-فردجداگانه مرتب‌سازی می‌شوند، که مبتنی بر روش مرتب سازی حبابی از مقایسه دو عنصر است و آنها را با قانون مقایسه تعویض می‌کند. اگر چه انواع الگوریتم‌های مرتب‌سازی شناخته شده بسیاری وجود دارد که می‌توان استفاده کرد، این روش تنها نیاز به n/2 تکرار از دو مرحله مرتب‌سازی دارد

۳-۳-۶-۲- گروه‌بندی زنبورها به کلونی‌های مختلف

نخ‌ها را در بلوک‌هایی به کلونی‌های مختلف طبق شناسه نخ آن‌ها تقسیم می‌کنیم، هر نخ به یک زنبور عسل و در جستجوی راه حل برای کلونی خود اختصاص داده می‌شود، به طوری که تعدادی از کلونی‌ها وجود دارند که الگوریتم زنبورها را به صورت موازی اجرا می‌کنند. تعداد زنبورها و کلونی‌ها در الگوریتم بستگی به این دارد که چه تعدادی از بلوک‌ها در شبکه و چه تعداد نخ‌ها در هر بلوک را تنظیم کنیم.

تعداد کلونی‌ها در یک بلوک = تعداد نخ‌ها در هر بلوک/تعداد زنبورها در هر کلونی.

۳-۳-۶-۳- تغییر الگوریتم زنبورها

برای موازی‌سازی BA استاندارد، برخی از مسائل پیاده‌سازی باید به شرح زیر در نظر گرفته شوند.

۳-۳-۶-۴- تغییر جستجوی محلی

جستجوی محلی در رویکرد BA سنتی، زنبورهای بیشتری (nep) برای مکان‌های نخبگان و زنبورهای کمتری (nsp) برای بقیه مکان‌ها از مکان‌های e بکارگیری می‌کند. این معقول است زیرا مکانیسم بر اساس احتمالات می‌باشد. اما در سیستم پیشنهادی، فقط زنبورهای nep برای بکارگیری در مکان‌های m برای تعادل بار در میان نخ‌ها اختصاص داده می‌شوند، برای دقت بیشتر، آن در معماری موازی حس نمی‌شود اگر برخی نخ‌ها پس از اتمام کار خود چیزی انجام ندهند و منتظر دیگران باشند.[۳۵].

۳-۳-۶-۵- دانه‌های تصادفی

می‌توانیم رشته‌های مختلف با دانه‌های تصادفی متفاوتی درGPU داشته باشیم. این کار می‌تواند با تولید موازی تعداد تصادفی از توالی‌های مختلف، به عملکرد کمک کند.

۳-۳-۶-۶- کاهش همسایگی

با توجه به روش جدید " کاهش همسایگی " در BA، ngh دائماً مقادیر را تغییر می‌دهد، در رویکرد پیشنهادی، جستجوهای محلی متعددی در مکان‌های مختلف به طور همزمان برای موازی‌سازی داریم، و زنبورهای بکارگیری شده در مکان‌های مختلف را با nghای مختلف در نظر می‌گیریم. تنظیم دیگر این است که نیازی نداریم تعداد زیادی زنبور استخدام‌شده مانند BA را تعیین کنیم ، چرا که کلونی‌های متعددی برای جستجوی همزمان داریم که این ریسک را به همراه داردکه ممکن است باعث کوچک‌سازی اشتباه کرد. در همین حال، کاهش سریع ngh می‌تواند زمان همگرایی سریعتر را به ارمغان آورد. معادله کاهش که استفاده می‌کنیم با معادله کاهش در الگوریتم زنبورها برابر است. در ابتدا، اندازه ngh به مقدار بزرگ تنظیم می‌شود.

۳-۳-۶-۷- ارتباطات با حافظه مشترک

در معماری‌های موازی به طور کلی ممکن است از حافظه مشترک یا روش عبور پیام برای برقراری ارتباط بین واحدهای پردازش متعدد استفاده شود. در معماری CUDA یک حافظه مشترک در همان بلوک وجود دارد، بنابراین از آن برای پیاده‌سازی ارتباطات در پایان هر تکرار استفاده می‌شود. در این استراتژی، سه مسئله برای نگرانی وجود دارد. اولین نگرانی این که چگونه اطلاعات به اشتراک گذاشته شود، دوم این است که با چه کسی به اشتراک گذاشته شود، و آخرین این است که چه مدت برای برقراری ارتباط در یک بار طول می‌کشد. مکانیسم‌های متعددی برای ارتباطات را مقایسه می‌کنیم. برای مثال، بهترین نتایج به دست آمده از کلونی‌های جداگانه در همان بلوک را پس از جستجوی همسایگی مرتب می‌کنیم، و بهترین را با دیگران به اشتراک می‌گذاریم. . به بیان دقیق‌تر ، مکان با کمترین برازندگی در هر کلونی با بالاترین برازندگی در بلوک جایگزین شده است. نتیجه نشان می‌دهد که نرخ همگرایی کاملاً خوب است. با این حال، روش مرتب‌سازی اغلب باعث تاثیر بر زمان اجرا می‌شود، در نهایت، ارتباط دو مرحله‌ای توسعه داده می‌شود که از مرتب‌سازی با نرخ همگرایی خوب، اجتناب می‌شود. تبادل زوج نیاز به زمان کمی برای به اشتراک گذاشتن دارد، و مرحله دوم همگرایی سراسری در طول زمان را بهبود می‌بخشد. روش ارتباطی در شکل(۳-۱۳) نشان داده شده است. دو نوع ارتباط به طور متناوب یکی پس دیگری اجرا شده‌اند[۳۵].

Adjacent exchange (first phase):

If *colony ID is odd*, then

exchange with colony (ID+1) % number of colonies per block

If *colony ID is even*, then

exchange with colony (ID-1) % number of colonies per block

Skip one exchange phase (second phase)

If *colony ID is odd*, then

exchange with colony (ID+2) % number of colonies per block

If *colony ID is even*, then

exchange with colony (ID-2) % number of colonies per block

شکل ۳-۱۲ : شبه کد مکانیزم ارتباطی در الگوریتم موازی زنبور در معماری کودا [۳۵]

۳-۳-۷- سخت افزار ارزیابی

در جدول(۳-۹) سخت افزاری که پیاده سازی الگوریتم موازی زنبور در آن به اجراء گذاشته شده است[۳۵].

جدول ۳-۹: سخت افزار مورد استفاده در پیاده سازی الگوریتم موازی زنبور [۳۵]

دستگاه	CPU	GPU
پردازنده	AMD Athlon(tm) II x4 disable 3 cores	GeForce GTX 460
تعداد هسته	4 Cores	336 cores (7 MPs)
سرعت پردازش	3.0 GHz	675 MHz
نوع حافظه	DDR3–1333	GDDR5
اندازه حافظه	4 GB	512 MB
سیستم عامل	Win7(32 bit)	----
ظرفیت محاسباتی	----	2.1
نسخه کودا	----	4.1

۳-۳-۸- توابع ارزیابی

در جدول(۳-۱۰) معادلات ۹ تابع پیوسته ارزیابی با معیار حداقل سازی نشان داده شده است[۳۵]. در معادلات محدوده های متغیرها نشان داده شده است.

جدول ۳-۱۰: توابع ارزیابی با محدوده مشخص بکار رفته در آنها [۳۵]

Function	Equation	Minimum
Ackley (2D)	$f(x1,x2) = 20 - 20e^{-0.2\sqrt{\frac{1}{2}(x1^2+x2^2)}} - e^{\frac{1}{2}(\cos(2\pi x1)+\cos(2\pi x2))} + e, -32 < x1 < x2$	$\vec{x} = (\vec{0}) f(\vec{x}) = 0$
Easom (2D)	$f(x_1,x_2) = -\cos(x_1) * \cos(x_2) e^{-(x_1-\pi)^2-(x_2-\pi)^2}, -100 < x_1 < 100$	$\vec{x} = (\pi,\pi) f(\vec{x}) = -1$
Goldstein and Price (2D)	$A(x_1,x_2) = 1 + (x_1+x_2+1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2) B(x_1,x_2)$ $= 30 + (2x_1+3x_2)^2(18 - 32x_1 + 12x_1^2 - 48x_2 + 36x_1x_2 + 27x_2^2) f(x_1,x_2) = AB, -2 < x_1 < s$	$\vec{x} = (0,-1) f(\vec{x}) = 3$
Martin and Gaddy (2D)	$f(x_1,x_2) = (x_1-x_2)^2 + \left[\frac{(x_1+x_2-10)}{3}\right]^2, -20 < x_i < 20$	$\vec{x} = (5,5) f(\vec{x}) = 0$
Schaffer (2D)	$f(x_1,x_2) = 0.5 + \frac{[\sin(\sqrt{x_1^2+x_2^2})]^2 - 0.5}{[1.0+0.001(x_1^2+x_1^2)]^2}, -100 < x_i < 100$	$\vec{x} = (\vec{0}) f(\vec{x}) = 0$
Schwefel (2D)	$f(x_1,x_2) = -x_1 \sin\left\|(\sqrt{\|x_1\|})\right\| - -x_2 \sin(\sqrt{\|x_2\|}), -500 < x_i < 500$	$\vec{x} = (420.97) f(\vec{x}) = -837.97$
Hyper Sphere (10D)	$f(\vec{x}) = \sum_{i=1}^{10} x_i^2, -100 < x_i < 100$	$\vec{x} = (\vec{0}) f(\vec{x}) = 0$
Griewank (10D)	$f(\vec{x}) = \frac{1}{4000}\sum_{i=1}^{10}(x_i-100)^2 - \prod_{i=1}^{10}\cos\left(\frac{x_i-100}{\sqrt{i+1}}\right) + 1, -600 < x_i < 600$	$\vec{x} = (10\vec{0}) f(\vec{x}) = 0$
Rosenbrock (10D)	$f(\vec{x}) = \Sigma_{i=1}^{9} 100(x_{i+1} - x_i^2)^2 + (1-x_i)^2, -50 < x_i < 50$	$\vec{x} = (\vec{0}) f(\vec{x}) = 0$

۳-۳-۹- پیاده سازی الگوریتم موازی زنبور

برای کوبا، ۵ پارامتر وجود دارد که باید تنظیم شود، GridDim، Block-Dim، N، M و NEP وجود دارد. در برنامه‌نویسی CUDA، کد در حال اجرای موازی به نام "هسته" است، اندازه کار هسته " گرید

" نامیده می‌شود. برنامه‌نویس باید تعداد ابعاد گرید را تنظیم کند. کوبا کار را به بسیاری از کارهای کوچکتر تقسیم خواهد کرد و آنها را به چند پردازنده‌های مختلف برای اجرا توزیع می‌کند. اندازه هر کار کوچکتر "بلاک" نامیده می‌شود. همچنین در تنظیم تعداد ابعاد گرید، برنام نویس باید تعداد ابعاد بلوک را تنظیم نماید، به این معنی که چه تعداد از نخ‌ها در یک بلوک هستند. در الگوریتم پیشنهادی، BlockDim/N کلونی در هر بلوک وجود دارد. به عنوان مثال اگر BlockDim = 256 و N = 8 را تنظیم کنیم، پس ۳۲ کلونی در یک بلوک وجود دارد. پارامترها با توجه به نتایج حاصل از آزمایش‌ها تنظیم می‌شوند، بعداً با جزئیات بیشتر بحث خواهیم کرد. تمام برنامه‌های BA و کوبا در آزمایش‌های زیر اجرا شده‌اند تا این که یا حداقل تابع به بهتر از ۰.۰۰۱ تقریب شود، یا به حداکثر تعداد چرخه دست یابد (در اینجا به ۵۰۰۰ تنظیم کرده‌ایم). در نسخه BA، چون تنها یک کاوش کلونی وجود دارد، اگر آن در کاهش ngh اشتباه کند، راه‌حل مطلوب سراسری هرگز یافت نمی‌شود. برای غلبه بر این، BA، nep و nsp را برای جلوگیری تا جایی که امکان پذیر است، کاملاً بزرگ تنظیم می‌کند. در حالت ایده‌آل، نسخه کوبا کلونی‌های موازی کاوش بسیاری در همان زمان دارد، بنابراین می‌توانیم موجب خطرات بیشتری شویم که باعث اشتباه در فرایند کاهش ngh می‌شود. برای اثبات این فرض، ۹ تابع با nep مختلف را آزمایش می‌کنیم، ۱، ۲، ۴، ۸، ۱۷ و ۳۲. در ابتدا GridDim = 4، BlockDim=256 تنظیم می‌کنیم. این تعداد مناسبی از BlockDim است. در بسیاری از GPUها، ۳۲ یا ۶۴ پردازنده جریان (کوچکترین واحد محاسبات) در چندپردازنده وجود دارد، بنابراین تعداد را عددهای متعددی از پردازنده‌های جریان در نظر می‌گیریم.هنگامی که nep سازگاری برای هر تابع یافتیم، سعی در کاهش تعداد BlockDim می‌کنیم، که بدان معنی است که تعداد کلونی‌ها کاهش یافته است. مسئله دیگر این است که چه اتفاقی خواهد افتاد اگر N و BlockDim را با همان عامل افزایش دهیم، به عبارت دیگر، زنبورها را برای هر کلونی افزایش دهیم و تعداد کلونی‌ها در یک بلوک را ثابت نگه داریم. درنهایتاً، از بهترین تنظیم پارامترهای یافت شده از سه آزمایش استفاده می‌کنیم، و نتیجه را با الگوریتم زنبورها مقایسه می‌کنیم[۳۵].

۳-۳-۹-۱- تحلیل nep

ابتدا nep را برای بررسی زمان اجرا و میزان تکرار قبل از همگرایی توابع معیار تغییر می‌دهیم. نتیجه در جدول ۳-۱۱ نشان داده شده است. برای توابع کم بعدی، تنها نیاز به nep بسیار کوچک (در حدود ۱ یا ۲) برای دریافت یک راه حل خوب با زمان کمتر داریم. اما برای توابع با ابعاد بالا، به nep بزرگتر نیاز داریم، nep بیش از حد کوچک منجر به همگرایی راه‌حل به تعداد با خطای بزرگ خواهد شد. علاوه بر این، راه‌حل با زمان اجرای کمتر به این معنا نیست که آن با تکرار کمتر است

۳-۳-۹-۲- تحلیل تعداد کلونی‌ها

دوم، تعداد کلونی‌ها را برای ارزیابی زمان اجرا و مقدار تکرار قبل از همگرایی توابع معیار تغییر می‌دهیم. نتیجه در جدول ۳-۱۲ نشان داده شده است، بسیاری از توابع عملکرد خوب و زمان اجرای کمتری با تعداد کمی از BlockDim، به استثنای سه تابع با ابعاد بالا، مانند HyperSphere (10D)، Griewank (10D)، و Rosenbrock(10D) دارند. برای این توابع با ابعاد بالا، تعداد کم از BlockDim همیشه مفید نیست، بهترین تعداد BlockDim برای HyperSphere و Griewank، ۱۲۸ و ۵۱۲ برای Rosenbrock می‌باشد. به طور مشابه، راه‌حل با زمان اجرای کمتر به این معنا نیست که در آن تکرار کمتر است.

جدول ۳-۱۱: افزایش با مقادیر nep [۳۵]

Benchmark functions	*nep* = 1		*nep* = 2		*nep* = 4		*nep* = 8		*nep* = 16		*nep* = 32	
	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations
Ackley(2D)	8.13	38	7.51	25	9.50	24	14.28	26	19.87	22	42.35	26
Easom(2D)	6.17	27	6.81	25	8.03	23	14.28	26	15.04	20	25.07	20
Goldstein and Price(2D)	6.01	22	5.56	12	7.57	17	9.14	14	12.92	13	18.49	11
Martin and Gaddy(2D)	4.58	17	4.72	16	4.92	14	5.67	15	5.52	8	5.74	5
Schaffer(2D)	7.11	29	7.27	22	9.24	22	11.81	19	16.20	16	33.28	20
Schwefel(2D)	5.52	27	6.35	30	6.48	22	8.33	23	11.18	21	16.20	19
HyperSphere (10D)	x	x	x	x	12	41	17.12	42	32.95	52	63.10	59
Griewank (10D)	x	x	52.51	51	70.95	43	125.23	44	219.76	42	439.90	44
Rosenbrock (10D)	x	x	x	x	x	x	1796.32	439	3140.61	410	4857.48	329

جدول ۳-۱۲: افزایش با مقادیر کلونی [۳۵]

Benchmark functions	*blockDim* = 32		*blockDim* = 64		*blockDim* = 128		*blockDim* = 256		*blockDim* = 512		*blockDim* = 768	
	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations
Ackley(2D)	6.89	40	6.05	31	6.76	32	8.13	38	8.341	27	12.66	30
Easom(2D)	x	X	5.27	31	6.02	32	6.28	29	7.76	26	10.88	25
Goldstein and Price (2D)	5.73	32	5.01	24	7.25	19	6.01	22	7.08	28	9.99	17
Martin and Gaddy (2D)	3.83	24	3.27	7	4.17	14	4.58	17	5.43	12	8.33	17
Schaffer(2D)	x	x	6.32	33	7.02	33	7.11	29	7.85	22	10.32	17
Schwefel (2D)	4.90	34	5.03	35	5.29	32	5.52	27	7.08	28	9.89	24
HyperSphere(10D)	19.71	67	22.36	56	15.79	42	17.12	42	20.15	41	25.45	41
Griewank (10D)	55.58	61	50.01	54	48.02	49	52.51	51	61.75	52	80.98	51
Rosenbrock (10D)	x	x	x	x	810.83	177	1796.32	439	432.984	94	598.962	102

۳-۳-۹-۳- تحلیل تعداد زنبورها

همچنین تعداد زنبورها را به منظور بررسی زمان اجرای و مقدار تکرار قبل از همگرایی توابع معیار تغییر می‌دهیم. جدول(۳-۱۳) نشان می‌دهد که می‌توانیم BlockDim و N را به تعداد کمتری در توابع با ابعاد کم تنظیم کنیم. در تابع با ابعاد بالا، برای برخی از توابع، می‌توانیم تعداد زنبورهای کمتری برای زمان اجرای کوتاه‌تر مانند HyperSphere و Griewank تنظیم نماییم. اما گاهی اوقات تعداد زنبورها نمی‌توانند بیش از حد کوچک باشد، یا راه‌حل با خطای بزرگی مانند Rosenbrock همگرا خواهد شد[۳۵].

۳-۳-۹-۴- پایداری و تسریع

زمان اجرا و میزان موفقیت پنجاه بار اجرا برای دو الگوریتم محاسبه شده است. برای تخمین زمان اجرای BA، پارامترهایی برای تمام توابع معیار در جدول(۳-۱۴) با توجه به مجموعه اصلی در مقاله [۳۷] داده شده است، و در جدول(۳-۱۵) پارامترهای تنظیم کوبا با استفاده از بهترین پارامترها که قبلاً یافتیم، نشان داده شده است.

جدول ۳-۱۳: بررسی زمان اجرا ء با افزایش با مقادیر تعداد زنبورها [۳۵]

Benchmark functions	*blockDim* = 128 *N* = 4		*blockDim* = 256 *N* = 8		*blockDim* = 512 *N* = 16	
	Time (ms)	Iterations	Time (ms)	Iterations	Time (ms)	Iterations
Ackley	6.76	36	8.13	38	9.68	33
Easom	5.72	31	6.28	29	7.87	26
Goldstein and price	5.35	23	6.01	22	7.13	17
Martin and Gaddy	4.11	22	6.01	22	6.00	19
Schaffer	6.76	18	6.35	23	8.72	26
Schwefel	4.70	26	5.52	27	7.61	30
HyperSphere	15.88	46	17.12	42	25.12	49
Griewank	53.62	57	52.51	51	65.10	53
Rosenbrock	x	x	1796.32	439	2166.75	463

جدول ۳-۱۴: بررسی مقادیر مناسب پارامترها برای اجرای الگوریتم زنبورها [۳۵]

Benchmark	*n*	*m*	*e*	*nep*	*nsp*	*stlim*
Ackley	30	8	1	20	10	5
Easom	20	14	1	30	5	10
GoldsteinAndPrice	10	4	2	30	10	10
MartinAndGaddy	10	7	1	30	10	10
Schaffer	10	4	2	30	10	10
Schwefel	20	14	1	30	5	10
HyperSphere	10	4	2	30	10	10
Griewank	20	18	1	10	5	5
Rosenbrock	10	4	2	30	10	10

جدول ۳-۱۵: بررسی مقادیر مناسب پارامترها برای اجرای الگوریتم زنبورها در پردازنده گرافیکی [۳۵]

Benchmark	*GridDim*	*BlockDim*	*n*	*m*	*nep*
Ackley	4	64	8	6	1
Easom	4	64	8	6	1
GoldsteinAndPrice	4	64	8	6	1
MartinAndGaddy	4	64	8	6	1
Schaffer	4	64	8	6	1
Schwefel	4	32	8	6	1
HyperSphere	4	128	8	6	8
Griewank	4	128	8	6	2
Rosenbrock	4	512	8	6	8

همچنین میزان موفقیت آن دسته از توابع با خطا و میزان موفقیت ۱۰۰٪ را با استفاده از هر دو الگوریتم در ۵۰ بار در الگوریتم زنبورهای استاندارد و BA مبتنی بر CUDA بررسی می‌کنیم. این نشان می‌دهد که کوبا می‌تواند به همان میزان در مقایسه با BA استاندارد به موفقیت دست یابد. در نهایت، تسریع از نظر زمان اجرا و تکرار اجرا شده بین آنها ارزیابی می‌شود. از آنجایی که CUDA از کتابخانه ریاضی سریع پشتیبانی می‌کند، ما را به استفاده از آنها نشویق می‌کند. نتایج ارزیابی در جدول(۳-۱۶) برای تسریع زمان اجرا و تسریع تکرار اجرا شده نشان داده شده است. این بدیهی است که نه تنها زمان اجرای کوبا کمتر از BA است، بلکه تکرار نیز کمتر اجرا شده است. نتیجه نشان می‌دهد که BA مبتنی بر CUDA ۱۳ تا ۵۶ بار سریع‌تر از BA در توابع معیار مختلف است. در نتیجه در جدول(۳-۱۷)، کوبا تکرار کمتری برای یافتن راه‌حل‌ها می‌گیرد. زمانی که توابع با ابعاد بالا را اجرا می‌کند، این باعث تفاوت زیادی می‌شود[۳۵].

جدول ۳-۱۶: بررسی شتاب برای اجرای الگوریتم موازی زنبورها در پردازنده گرافیکی [۳۵]

Benchmark Functions	Standard BA	CUDA-based BA(CUBA)	Speedup
Ackley	273 ms	6.05 ms	45.12
Easom	70 ms	5.27 ms	13.28
Goldstein and Price	92 ms	5.01 ms	18.36
Martin and Gaddy	76 ms	3.27 ms	27.24
Schaffer	231 ms	6.32 ms	36.55
Schwefel	279 ms	4.90 ms	56.93
HyperSphere	389 ms	15.79 ms	24.63
Griewank	2520 ms	48.02 ms	52.47
Rosenbrock	5595 ms	432.98 ms	12.92

جدول ۳-۱۷: بررسی شتاب زمان اجرا و تعداد تکرار برای اجرای الگوریتم موازی زنبورها در پردازنده گرافیکی [۳۵]

Benchmark Functions	Standard BA	CUDA-based BA (CUBA)	Iteration reduction factor
Ackley	90	31	2.90
Easom	44	31	1.41
Goldstein and Price	49	24	2.04
Martin and Gaddy	50	7	7.14
Schaffer	55	33	1.66
Schwefel	141	34	4.14
HyperSphere	158	42	3.76
Griewank	1487	49	30.34
Rosenbrock	4456	94	47.40

در این بخش، ابتدا الگوریتم زنبورهای موازی بر اساس CUDA ارائه شد. روش جستجوی محلی، اجرا در معماری سخت‌افزار SIMT (یک دستورالعمل چند نخ) را تغییر دادیم و دو بخش از مکان‌های جستجوی محلی را برای اجتناب از به هدر رفتن قدرت محاسبات GPU ادغام کردیم. به همین دلیل، ما هیچ روش رها کردن مکان نداریم. علاوه بر این، در نظر می‌گیریم، زنبورهای بکارگیری شده در مکان‌های مختلف ngh خود را حفظ می‌کنند، به این معنی که آنها به طور مستقل کوچک‌سازی می‌کنند. کلونی‌ها را در همان بلوک به طور جداگانه با استفاده از الگوریتم مرتب‌سازی زوج-فرد برای دریافت سود از موازی مرتب‌سازی می‌کنیم. مکانیسم ارتباط بین کلونی‌ها در همان بلوک نیز نقطه‌ی مهمی برای کاهش زمان همگرایی است، در الگوریتم پیشنهادی، ارتباط دو مرحله‌ای برای نتیجه بهتر را انتخاب می‌کنیم.برای یافتن ویژگی‌های این الگوریتم جدید، پارامترها را تغییر دادیم، و با استفاده از nep کوچک، در نتیجه بسیاری از توابع با ابعاد کم می‌توانند با عملکرد خوب اجرا شوند. این یکی از نکات کلیدی است که چرا کوبا با زمان همگرایی سریعتری از الگوریتم زنبورهای استاندارد اجرا می‌شود. همچنین سعی می‌کنیم تعداد کلونی‌ها در یک بلوک CUDA را کاهش دهیم و تعداد زنبورها در هر کلونی را برای بهینه‌سازی پارامترهای تعیین شده برای هر عملکرد کاهش می‌دهیم. همچنین زمان همگرایی (خطا کمتر از ۰.۰۰۱) الگوریتم زنبورهای CUDA با الگوریتم زنبورها مقایسه شده است. نتیجه تجربی نشان می‌دهد کوبا از BA حداقل ۱۳ بار در ۹ تابع مختلف از مسائل بهینه‌سازی سریع‌تر است. در آینده، کوبا را با دیگر الگوریتم‌های مبتنی بر ازدحام موازی مقایسه خواهیم کرد و سعی می‌کنیم الگوریتم مرتب‌سازی موازی‌تر و مکانیزم ارتباطات را ارائه نماییم. نه تنها برای حل مسئله بهینه‌سازی، بلکه الگوریتم پیشنهادی را در برنامه‌های کاربردی دنیای واقعی اعمال می‌کنیم. امروزه، محاسبات ابری بسیار مهم‌تر و محبوب‌تر شده است. برخی بسترهایی وجود دارند که ارائه دهنده سرویس‌های محاسبات ابری مبتنی بر GPU می‌باشند. الگوریتم پیشنهادی را بهبود می‌بخشیم و در محیط خوشه‌های GPU آزمایش می‌کنیم[۳۵].

۳-۴- سایر پژوهش‌ها

رابینویچ[1] [۴۱] جهت افزایش شتاب الگوریتم ذرات استاندارد، آن را در پردازنده گرافیکی[2] به اجراء گذاشت، به شکلی که در این الگوریتم ذرات به گروه‌های مستقل از هم تقسیم شده و هر دسته در قسمتی از پردازنده گرافیکی به اجراء گذاشته می‌شود. در این روش در هر تکرار هر دسته یک جواب مناسب پیدا می‌کند و در هر مرحله جواب های دسته ها را با هم مقایسه می‌شوند تا بهترین ذره سراسری یافت شود، از پیشنهادات این مقاله استفاده از توزیع تصادفی K جهت دسته بندی ذرات

[1] rabinovich
[2] GPU

می‌باشد. نتایج ارزیابی ها و شبیه سازی روی تابع ارزیابی نشان می‌دهد که در سیستمی با پردازنده گرافیکی GTX 456 سرعت الگوریتم پیشنهادی و موازی ۵ برابر شده است. برای داشتن دقت در جواب ها توصیه شده است که ذرات از شبکه عصبی برای یادگیری استفاده نمایند.

وان لونگ[۱] [۴۲] نشان داد الگوریتم های متاهیورستیک محلی[۲] روش های موثری برای حل مسائل پیچیده صنعتی هستند. این الگوریتم‌ها باعث می‌شوند که اندازه فضای جستجو و زمان جستجو کاهش یابد. وقتی داده‌های ما حجم زیادی دارند برای افزایش سرعت بهتر است در پردازنده گرافیکی اجراء شود. نتایج نشان می‌دهد که اجرای الگوریتم های جستجوی محلی در پردازنده گرافیکی متوسط عملکردی با سرعت ۸۰ برابر را نشان می‌دهد.

آرورا[۳] [۴۳] در پژوهش خود جهت موازی سازی الگوریتم ژنتیک از توابع کتابخانه‌ای[۴] تحت چارچوب کودا استفاده کرده است. او پیشنهاد داده است که فازهای الگوریتم ژنتیک را در پردازنده گرافیکی اجرا شود و به این منظور باید کرنل‌های مختلفی جهت انتساب مقادیر اولیه به پارامترها، جهش و ترکیب ایجاد شود. فرق اصلی این نوع پیاده سازی با الگوریتم ژنتیک استاندارد در این است که جمعیت اولیه و جمعیت های تحت جهش و ترکیب همه در پردازنده گرافیکی قرار دارند در واقع هر هسته می‌تواند یک ژن و هر بلاک می تواند یک کروموزم باشد.

الگوریتم کرم شب تاب یک الگوریتم هوش ازدحامی است که با الهام از رفتار نورپراکنی کرم های شب تاب عمل می‌نماید. این الگوریتم برای حل مسائل غیر خطی و پیچیده بکار می‌رود. این الگوریتم در اکثر مواقع جواب مناسبی پیدا می‌کند. یکی از ایرادات این الگوریتم این است که باید به اندازه کافی تکرار شود تا به جواب های دقیق همگرا شود. هیوسلمن[۵] و هاویچ[۶] در [۴۴] جهت موازی سازی این الگوریتم در کودا، بردارهای موقعیت و سایر بردارهای الگوریتم کرم شب تاب را با استفاده از اشاره گر به آرایه به پردازنده گرافیکی منتقل کردند و برای به روز رسانی و تغیرات کرم‌های شب تاب در فضای مسئله از کرنل‌های قابل اجراء در پردازنده گرافیکی استفاده کردند. این پژوهش نشان می‌دهد که سرعت اجرای الگوریتم موازی کرم شب تاب در پردازنده گرافیکی در این روش نسبت به پردازنده اصلی تا ۳۹ برابر بیشتر شده است.

گاردناز[۷] در [۴۵] تحقیق خود را روی الگوریتم ذرات را به شیوه تقسیم و غلبه در واحد پردازش کارت گرافیک به اجراء گذاشت. در این روش جمعیت اولیه به چند بخش تقسیم می‌شود و هر بخش را در یک قسمتی از پردازنده کارت گرافیک اجرا می‌شود و روی هر بخش الگوریتم ذرات اجرا می‌شود. سرعت پیدا کردن جواب ها در این روش به شکل مطلوبی بهبود می‌یابد.

---

[1] Van luong
[2] LSMS
[3] arora
[4] API
[5] husselman
[6] hawich
[7] gardenaz

کالازان [1] [۴۶] سه پیاده سازی موازی الگوریتم ذرات یعنی PPSO، SGPSO و PDPSO پیاده سازی و مقایسه کرده است. در روش اول ذرات موازی سازی شده‌اند و در روش دوم، الگوریتم فضای جستجو به زیرفضاهایی تقسیم می‌کند و در پیاده سازی سوم ابعاد و بعدهای مسئله در پردازنده گرافیکی اجراء شده و هر نخ[2] یک بعد از مسئله را مدیریت می‌کند. نتیجه گیری مقاله با توجه به نمودار ها نشان می‌دهد که الگوریتم PDPSO نسبت به مابقی بهتر عمل می‌کند و هر چه تعداد داده ها بیشتر باشد اختلاف این الگوریتم با دو الگوریتم دیگر بیشتر خواهد شد. نتایج نشان می‌دهد هر بعد با یک نخ پیاده سازی شود جواب بهتری حاصل می‌شود.

هیونگ[3] و وانگ[4][۴۷] در جهت تبادل اطلاعات بین ذرات در الگوریتم ذرات از حافظه اشتراکی[5] استفاده کردند. نتایج الگوریتم پیشنهادی آن‌ها در یکی از آزمایشات نشان داد که اگر تعداد داده ها را ۶۵۵۳۶ ذره در نظر گرفته شود و ابعاد مسئله ۱۰۰ بعدی فرض شود و کارت گرافیک از نوع NVIDIA Tesla C1060 1.30 GHz باشد سرعت و شتاب در ۲۸۰ برابر می‌شود.

ساتو[6]در [۴۸] تحقیقات خود توانسته مسئله کلاسیک جدول سودکو را توسط الگوریتم موازی ژنتیک در کودا حل کند. حل جدول سودکو یک مسئله دشوار است. برای حل این مسئله تکنیک های مختلفی از جمله ژنتیک بکار می‌رود اما ایراد الگوریتم ژنتیک زمان اجرایی بالای این الگوریتم است. در این پژوهش تلاش شده است که الگوریتم ژنتیک برای حل مسئله سودوکو را تحت پردازنده گرافیکی به شکل موازی اجراء شود. نتایج شبیه‌سازی در کارت گرافیک NVIDIA GTX 460 برای جدول های مختلف با پیچیدگی مختلف نشان می‌دهد که سرعت از ۱۶ برابر تا ۲۵ برابر شد و دقت حل مسئله نیز ۱۰۰ درصد بدست آمد.

۳-۵- جمع بندی

در این فصل موازی سازی دو الگوریتم تکاملی هوش دسته جمعی ماهی و زنبور عسل در معماری کودا با جزئیات مورد بررسی قرار گرفته شد تا خواننده حداقل با چند روش مطالعاتی در زمینه موازی سازی الگوریتم‌های تکاملی در معماری کودا آشنایی داشته باشد. بررسی مقالات و مطالعات مرتبط می‌تواند ایده‌های لازم برای پیاده‌سازی روش پیشنهادی را فراهم آورد در فصل بعدی جزئیات روش پیشنهادی برای موازی‌سازی الگوریتم تکاملی گرده‌افشانی گل تشریح خواهد شد.

[1] calazan
[2] tread
[3] hung
[4] vang
[5] Shared memory
[6] sato

فصل ۴- روش پیشنهادی

۴-۱- مقدمه

اکثر مسائل موجود و صنعت به گونه‌ای است که پژوهشگران علاقمند می‌باشند که بهترین طرحها را با حداقل هزینه‌ها[۱] پیاده‌سازی نمایند. به عبارت دیگر بسیاری از مسائل پیرامون ما در قالب مسائل بهینه‌سازی مدلسازی می‌شوند که دارای راه‌حلهای مختلفی از نظر میزان بهینه بودن[۲] می‌باشند. یک مسئله بهینه‌سازی عموماً به شکل روابط و معادلات ریاضی و در قالب تابع هدف[۳] ارایه می‌شود که هدف نهایی یافتن بیشینه یا کمینه این تابع بسته به ماهیت مسئله می‌باشد. مسائل بهینه‌سازی واقعی به طور معمول پیچیده و غیرخطی[۴] می‌باشند که حل آنها را توسط روش‌های ریاضی مانند گرادیان[۵] و سیمپلکس[۶] سخت و تا حدی غیر ممکن می‌نماید. روش‌های تکاملی نوع جدیدی از محاسبات تکاملی جهت حل مسائل سخت و دشوار بهینه‌سازی محسوب می‌شوند که روز به روز به تعداد آنها افزوده می‌شود. در الگوریتم‌های تکاملی[۷] یک پدیده طبیعی و تکاملی که بیشتر ماهیت زیستی دارد دست مایه تقلید قرار گرفته تا به کمک این مکانیزم موجود در طبیعت و شیوه تکاملی آن، یک مسئله دشوار به سهولت حل شود. حل مسائل بهینه‌سازی به کمک روش‌های تکاملی که قادرند طیف وسیعی از مسائل کاربردی را حل نمایند نشان دهنده اهمیت موضوع می‌باشد. امروزه پدیده‌های مختلفی در طبیعت وجود دارد که می‌تواند در حل مسائل سخت و دشوار به پژوهشگران کمک فراوانی نماید لذا ایجاد الگوریتم‌های تکاملی جدید نظیر گرده‌افشانی گلها و بهبود دقت و سرعت آنها موضوع بسیاری از پژوهش‌ها می‌باشد. یکی از دلیل این پژوهش را می‌توان بهبود سرعت الگوریتم تکاملی گرده‌افشانی گل برای حل مسائل مختلف بهینه‌سازی در نظر گفت که این پژوهش اولین تلاش در این راستا می‌باشد. الگوریتم‌های تکاملی[۸] یکی از شیوه‌های موثر در حل مسائل بهینه‌سازی[۹] می‌باشند که از سازوکارهای طبیعی و روابط حاکم بر آن الهام گرفته شده‌اند. تکامل گیاهان در طول تاریخ پیدایش آنها موضوعی جالب برای خلق الگوریتم‌های تکاملی است. گیاهان ابتدایی قادر به تولید گل به عنوان اندام جنسی نبودند و به مرور زمان با تکامل و انتخاب طبیعی[۱۰] دارای این اندام برای تولید مثل بهینه‌تر شدند. امروزه گونه‌های گلدار بخش وسیعی از گیاهان را شامل می‌شود به گونه‌ای که تولید مثل آنها بیشتر بر اساس مکانیزم گرده‌افشانی صورت می‌پذیرد. در طبیعت گل‌های که موفق به گرده‌افشانی می‌شوند دوام بیشتری دارند که در نهایت این گل‌ها به میوه و دانه که گیاهان آینده را ایجاد می‌نمایند، تبدیل

---

[1] Cost
[2] Optimal
[3] Objective function
[4] Nonlinear
[5] Gradient
[6] Simplex
[7] Evolutionary algorithms
[8] evolutionary algorithms
[9] optimization problems
[10] natural selection

می‌شوند. تکامل گونه‌ها به شکلی است که موجودات شایسته[1] شانس بیشتری برای بقاء دارند که گیاهان و گل‌ها از این قائده مستثنی نمی‌باشند. الگوریتم گرده‌افشانی گل‌ها[2] یک الگوریتم تکاملی مبتنی بر رفتار گیاهان و مکانیزم گل‌دهی آنها می‌باشد. در این الگوریتم راه‌حل‌های مسئله در قالب گل‌ها مدلسازی شده و فرض می‌شود که هر گیاه قادر به تولید یک گل و هر گل نیز یک گرده تولید می‌نماید. در این الگوریتم، گل‌ها دو نوع گرده‌افشانی سراسری[3] و محلی[4] را انجام می‌دهند که در حالت گرده‌افشانی سراسری فرض بر این است که گرده‌ها با یک توزیع احتمالی توسط حشرات در فضای جستجوی[5] مسئله پرواز می‌نمایند. الگوریتم‌های تکاملی برای رسیدن به جوابهای بهینه[6] و همگرایی نیاز به تعداد قابل توجه‌ای جمعیت اولیه و تکرار دارند که این حالت باعث افزایش زمان اجرای الگوریتم گرده‌افشانی گل‌ها می‌شود. الگوریتم گرده‌افشانی گل مانند هر الگوریتم تکاملی ماهیتی موازی دارد بطوریکه که هر کدام از اعضای جمعیت می‌توانند به شکل مستقل فضای جستجوی مسئله را مورد جستجو قرار دهد. تعداد هسته‌های پردازنده اصلی محدود است که جهت اجرای موازی هر کدام از اعضای جمعیت، این معماری مناسب نیست و بیشتر برای حالتی که اعضای جمعیت به شکل سریالی در پردازنده اصلی اجراء می‌شوند مناسب می‌باشد. معماری پردازنده گرافیکی[7] بر خلاف پردازنده اصلی[8] دارای هسته‌های بیشتری است که برای اجرای موازی اعضای جمعیت اولیه بسیار مناسب‌تر می‌باشد و از طرفی ایجاد صف در این پردازنده نسبت به پردازنده اصلی کمتر است. یکی دیگر از ویژگی‌های ممتاز پردازنده گرافیکی اختصاصی بودن آن در مقابل عمومی بودن پردازنده اصلی است. به عبارت ساده‌تر پردازنده اصلی در اختیار تعداد زیادی پروسه مانند پروسه‌های سیستم عامل و الگوریتم تکاملی است و این در حالی است که پردازنده گرافیکی فقط می‌تواند در اختیار برنامه اختصاصی نظیر الگوریتم تکاملی قرار بگیرد. چارچوب کودا[9] یک کتابخانه از دستورات موازی را در اختیار برنامه‌نویسان پردازنده گرافیکی قرار می‌دهد تا برنامه‌های موازی خود را به شکل موازی به اجراء بگذارند. الگوریتم گرده‌افشانی گل دارای اعضای جمعیتی است که در این پژوهش سعی می‌شود هر گل را با یک نخ[10] و به شکل موازی در این معماری موازی سازی نمایم. در این پژوهش جهت موازی‌سازی از کتابخانه فرعی کودافای[11] جهت موازی سازی الگوریتم گرده‌افشانی گل‌ها استفاده می‌نمایم و سعی می‌شود یک نسخه موازی از این الگوریتم به کمک این معماری توسعه

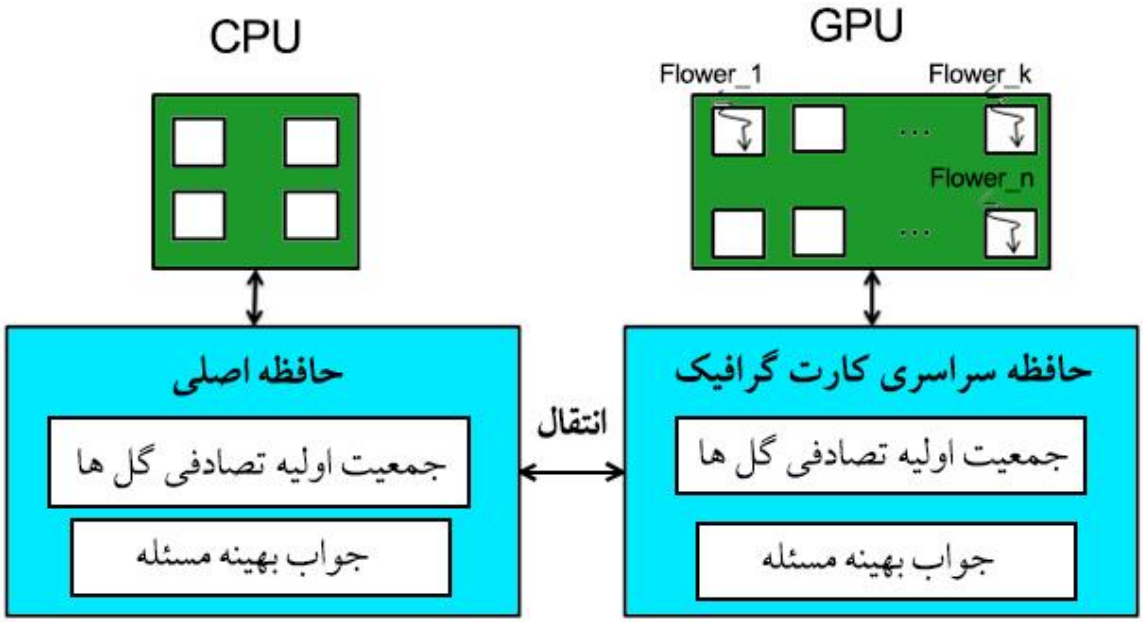
---

[1] fitness
[2] flower pollination algorithm
[3] cross pollination
[4] self pollination
[5] search space
[6] optimal
[7] graphics processing unit (GPU)
[8] central processing unit (CPU)
[9] CUDA
[10] Threads
[11] Cudafy.net

داده شود. در این فصل چارچوب روش پیشنهادی به همراه مراحل روش پیشنهادی تشریح خواهد شد.

۴-۲- چارچوب روش پیشنهادی

در شکل(۴-۱)، یک چارچوب مفهومی از روش پیشنهادی ارایه شده است این چارچوب دارای سطح انتزاعی بالایی است که نشان می‌دهد جمعیت اولیه که همان راه‌حلهای اولیه مسئله می‌باشند به شکل تصادفی درون حافظه پردازنده اصلی(میزبان) ایجاد شده است سپس این جمعیت اولیه از گلها از طریق گذرگاه به حافظه سراسری کارت گرافیک(دستگاه) کپی می‌شود. جمعیت اولیه‌ای که درون حافظه سراسری دستگاه قرار گرفته است به شکل یک اشاره‌گر از نوع آرایه تعریف می‌شود و همان طور که قبلا اشاره شد، از آنجایی که پردازنده کارت گرافیک دارای هسته های محاسباتی زیادی می باشد ، هر چند خانه از این آرایه به عنوان یک عضو جمعیت اولیه در یک هسته و توسط یک نخ مدیریت می‌شود. نخ‌هایی که گل‌ها(اعضای جمعیت اولیه) را زمانبندی می‌نمایند یک تابع واحد(کرنل) را که همان دستورات الگوریتم گرده‌افشانی گل است را بر روی این جمعیت پیاده‌سازی می‌نمایند. در این چارچوب بعد از اجرای الگوریتم پیشنهادی یک گل که بهترین ارزیابی را در بین تمام گام‌های الگوریتم داشته است به عنوان بهترین عضو جمعیت سراسری انتخاب شده و از حافظه دستگاه به حافظه میزبان منتقل می‌شود. در ادامه این فصل مراحل و جزئیات روش پیشنهادی تشریح خواهد شد.



شکل ۴-۱ : چارچوب کلی روش پیشنهادی

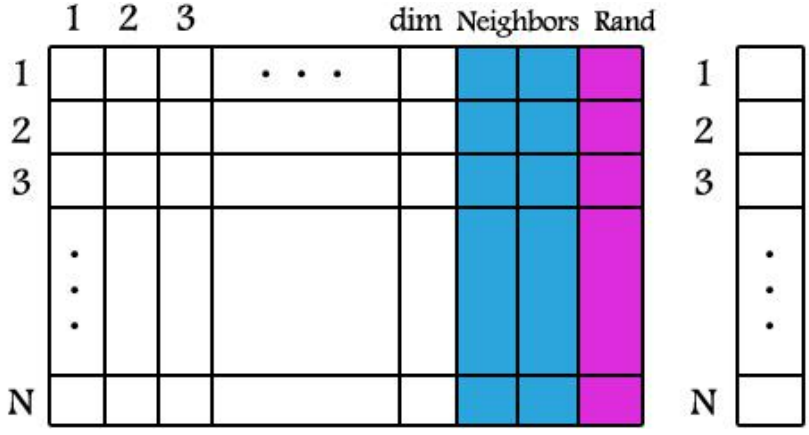
۴-۳- الگوریتم پیشنهادی

برای کاهش زمان اجراء و افزایش سرعت الگوریتم گرده‌افشانی گل توسط معماری کودا نیاز است که بخش های مهم این الگوریتم که محاسبات در آنجا صورت می‌پذیرد به شکل موازی اجـراء شـود. در این بخش مراحل مختلف روش پیشنهادی تشرح می‌شود.

۴-۳-۱- اعضای جمعیت اولیه

راه‌حل‌های اولیه هر مسئله بهینه‌سازی در قالب اعضای جمعیت اولیه شناخته می‌شوند و نیازاست که به درستی مدل‌سازی شوند. فرض کنید که اعضای جمعیت اولیه به شکل مجموعه $\boldsymbol{X} = \{\boldsymbol{P_1}, \boldsymbol{P_2}, \dots, \boldsymbol{P_n}\}$ تعریف می‌شوند و به فرض اینکه فضای مسئله D بعدی باشد عضو i ام جمعیت بردار $\boldsymbol{x_i} = \ll \boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_D} \gg$ خواهد بود. بنابراین ما یک آرایه N در D بعدی خواهیم داشت که N تعداد جمعیت اولیه و D تعداد ابعاد مسئله می باشد

اما ما برای اجرا در GPU به جمعیت اولیه، دو متغیر Neighbor و یک rand را اضافه می کنیم تا نیازی به تولید اعداد تصادفی توسط GPU نباشد و بتوان مستقیما از این متغیرها برای محاسبات استفاده کرد :



شکل ۴-۲ : مدلسازی و کدینگ جمعیت اولیه

همچنین مقدار شایستگی هر یک از اعضای جمعیت اولیه را در آرایه ای N بعدی ذخیره می کنیم.

۴-۳-۲- تخصیص حافظه برای جمعیت اولیه

جمعیت اولیه در الگوریتم پیشنهادی به شکل مجموعه‌ای از گل‌ها در فضای مسئله وجود دارد. قبل از تولید تصادفی این جمعیت‌ها نیاز است که میزان فضای حافظه آنها در حافظه سراسری دستگاه(کارت گرافیک) مشخص باشد. رزرو حافظه در کودا به کمک مجموعه دستورات زیر صورت می‌پذیرد:

```
float *d_Values;
cudaMallocPitch(&d_Values ,&pitch ,dim * sizeof(float), N);

float *d_Fitness;
cudaMalloc( (void**)&d_Fitness, N * sizeof(float));
```

در این مجموعه دستورات، متغیر d_Values مکان و موقعیت جمعیت اولیه و d_Fitness میزان شایستگی هر یک از اعضای جمعیت اولیه را درون حافظه سراسری دستگاه یا کارت گرافیک نشان می دهد.

در کودا از دستور cudaMalloc برای تخصیص فضای یک بعدی و از cudaMallocPitch برای تخصیص فضای دو بعدی در حافظه سراسری کارت گرافیک استفاده می شود. اختصاص فضا به متغیرهای کودا قبل از مقدار دهی اولیه باعث می‌شود در حین اجرای برنامه متغیری با کمبود حافظه مواجه نشود.

۴-۳-۳- متغیرهای مهم الگوریتم

از آنجایی که تولید جمعیت اولیه توسط CPU انجام میگیرد بنابراین نیاز به آرایه ای مشابه با d_Values در حافظه اصلی داریم:

```
float Values[N][dim];
```

پس از تولید جمعیت اولیه می توان آن را به d_Values در GPU منتقل کرد.

پس از پایان اجرای الگوریتم توسط GPU، لازم است جواب بهینه و مختصات آن نمایش داده شود، و از آنجایی که امکان دسترسی مستقیم به مقادیر متغیرهای موجود در حافظه GPU وجود ندارد، باید یک سری متغیر متناظر هم در میزبان و هم دستگاه ایجاد شود:

```
float *d_min;
cudaMalloc( (void**)&d_min, 1 * sizeof(float) );
float *d_minPosition;
cudaMalloc( (void**)&d_minPosition, dim * sizeof(float) );

float min[dim];
float minPosition[1];
```

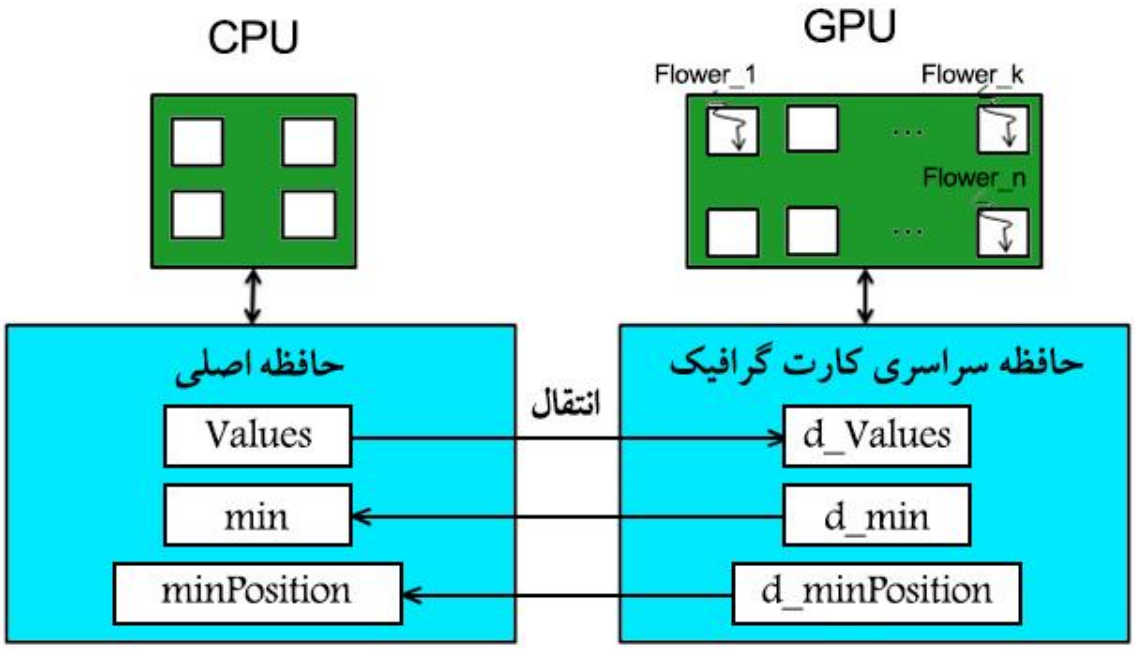
d_min و d_minPosition به ترتیب مقدار شایستگی بهترین جواب مسئله و مختصات آن در حافظه سراسری کارت گرافیک و min و minPosition معادل آن در حافظه اصلی سیستم می باشد.

که توسط دستور زیر عمل انتقال از حافظه GPU به حافظه اصلی صورت می گیرد:

```
cudaMemcpy( Device , Host, Size * sizeof(float),
cudaMemcpyHostToDevice );
```

که **Device** متغیر واقع در حافظه GPU و **Host** متغیر واقع در حافظه اصلی دستگاه و **Size** طول آرایه های متناظر می باشد

در شکل(۴-۳) موقعیت متغیرهای اصلی نشان داده شده است:



شکل ۴-۳ : موقعیت متغیرهای اصلی در حافظه سراسری GPU و حافظه اصلی سیستم

۴-۳-۴- انتساب هر عضو جمعیت(گل) به مجموعه نخ‌ها

در روش الگوریتم استاندارد گرده‌افشانی گل، جمعیت اولیه گل‌ها در حافظه میزبان به حالت یک ماتریس $N \times (D+1)$ تعریف می‌شود که ساختار این جمعیت اولیه در حافظه سراسری دستگاه، یک آرایه $n \times D$ بعدی و یک آرایه N بعدی است. در روش پیشنهادی هر نخ داده های یک سطر از d_Values که معادل یک گل می باشد را مدیریت می کند و پس از بروز رسانی مقادیر، مقدار شایستگی جدید را خانه متناظر در d_Fitness ذخیره میکند. در این روش هر نخ یک نسخه از الگوریتم گرده‌افشانی گل‌ها را بر روی داده‌های گل اعمال می‌نماید. برنامه‌ای که نخ‌های مختلف بر روی گل‌های موجود در آرایه d_Values اجراء می‌نمایند یک تابع است که شامل دستورات واحدی برای نخ‌های متفاوت است. تابعی که توسط نخ‌ها به اجراء گذاشته می‌شود در ادبیات موازی‌سازی کودا کرنل[1] نامیده می‌شود.

هر نخ در برنامه‌نویسی کودا به یک شماره منحصربفرد نیاز دارد که توسط کودا شناسایی شود. شناسایی توسط کودا باعث می‌شود هر نخ به درستی داده‌های مسئله را پردازش نماید.

[1] Kernel

۴-۳-۵- کرنل

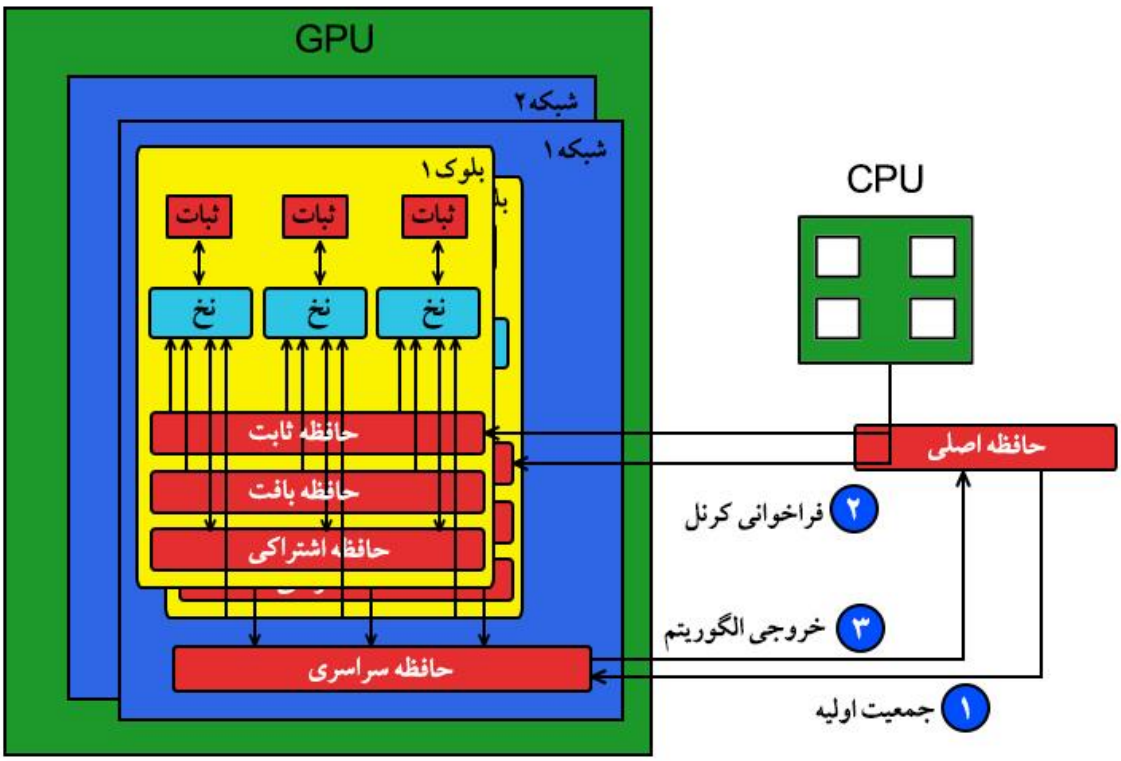
عامل اصلی افزایش سرعت اجرای الگوریتم هایی که توسط GPU اجرا می شوند، کرنل می باشد و طراحی مناسب کرنل ها می تواند موجب تسریع بیشتر اجرای الگوریتم شود، کرنل در حقیقت همان function یا تابع در زبان های برنامه سازی است که توسط CPU فراخوانی شده و توسط GPU اجرا می شود. به طور کلی کرنل‌ها فقط قابلیت اجراء شدن بر روی هسته‌های پردازنده گرافیکی را دارند و هر نخ یک نسخه و یک کپی از این کرنل را بر روی داده‌های خود(گل) اعمال می‌نماید

ساختار کلی یک کرنل به صورت زیر می باشد :

```
__global__ void kernel(float * d_Values ,float *d_ Fitness)
{
}
```

همانطور که در بالا مشاهده می شود درج کلید واژه **__global__** قبل از تابع الزامی است، همچنین در تعریف کرنل، خروجی مستقیم وجود ندارد و حتما باید خروجی از نوع void تعریف شود، هر چند به دلیل این که ورودی های تابع از نوع اشاره گر می باشند نیازی به خروجی وجود ندارد و هر تغییری روی متغیرهای ورودی موجب بروز رسانی متغیرها می شود.

می‌توان فراخوانی و اجرای کرنل را به شکل(۴-۴) و در قالب یک دیاگرام نمایش داد:



شکل ۴-۴ : مراحل فراخوانی و اجرای کرنل

مراحل این دیاگرام به شرح زیر است :

۱- ابتدا جمعیت اولیه توسط CPU تولید شده و به حافظه سراسری GPU منتقل می شود
۲- بسته به تعداد بلوک هایی که باید الگوریتم را اجرا کنند، کرنل توسط CPU از حافظه اصلی به حافظه ثابت هر یک از بلوک های فعال کپی شده و توسط هر یک از نخ ها اجرا می شود
۳- پس از اتمام اجرای الگوریتم، خروجی از حافظه سراسری GPU به حافظه اصلی سیستم منتقل شده و قابل دسترسی می باشد.

جهت فراخوانی یک کرنل در کودا از دستور زیر استفاده می کنیم:

```
kernel<<<dimGrid,dimBlock>>>( d_Values, d_Fitness);
```

همانطور که در فصول قبل اشاره شد، GPU از تعدادی شبکه تشکیل شده، هر شبکه به تعدادی بلوک تقسیم شده و هر بلوک دارای تعدادی نخ می باشد که وظیفه اجرای کرنل را به عهده دارند.
dimGrid مشخص کننده تعداد بلاک های به خدمت گرفته شده از هر شبکه و dimBlock مشخص کننده تعداد نخ های به خدمت گرفته شده از هر بلوک می باشد.
در پردازنده های سری Geforce، حداکثر تعداد نخ های به خدمت گرفته شده برای اجرای الگوریتم را ۵۱۲ عدد در نظر گرفته است که این عدد در سری های مختلف متفاوت است.
در نهایت می توان با دستور زیر حافظه اختصاص داده شده به متغیرها را به کارت گرافیک برگرداند.

```
cudaFree( نام متغیر موجود در کارت گرافیک );
```

۴-۳-۵-۱- کرنل محاسبه شایستگی

برای محاسبه شایستگی هر عضو جمعیت(گل)، می توان مدیریت هر عضو را بر عهده یک نخ قرار داد.
هر نخ پس از محاسبه شایستگی، مقدار مورد نظر را در خانه متناظر در آرایه d_ Fitness قرار می دهد.

شبه کد کرنل محاسبه شایستگی:

شاخص سراسری نخ ها = i
محاسبه شایستگی سطر i ام آرایه d_Values
قرار دادن مقدار شایستگی در خانه i ام آرایه d_ Fitness یا d_ Fitness[i]

۴-۳-۵-۲- کرنل محاسبه شایستگی بهینه (کمترین یا بیشترین)

برای محاسبه کمترین یا بیشترین مقدار شایستگی محاسبه شده، آرایه d_ Fitness را به دو آرایه A و B تقسیم می کنیم، سپس دو به دو خانه های متناظر دو آرایه را با هم مقایسه کرده، مقدار بهینه را در خانه متناظر A قرار می دهیم، پس از پایان مرحله اول، آرایه A حاوی مقادیر بهینه دو آرایه A و B می باشد

حال می توان الگوریتم بالا را بر روی آرایه A اجرا کرد، شرط پایان الگوریتم زمانی است که تعداد خانه های آرایه A کمتر از ۲ باشد

شبه کد کرنل محاسبه شایستگی بهینه:

تقسیم آرایه d_ Fitness به دو آرایه A و B

تا زمانیکه اندازه آرایه A بزرگتر از ۲ بود

شاخص سراسری نخ ها = i

مقایسه مقادیر A[i] و B[i] و قرار دادن مقدار بهینه در A[i]

تقسیم آرایه A[i] به دو آرایه A و B

۴-۳-۵-۳- کرنل بروز رسانی موقعیت هر عضو جمعیت (گل)

در الگوریتم استاندارد گرده افشانی گل، به دفعاتی مشخص (iteration) عمل بروز رسانی تک تک اعضای جمعیت را انجام می دهیم که در این صورت زمان اجرای الگوریتم، ضریبی از iteration × N خواهد بود.

در الگوریتم پیشنهادی، بروز رسانی هر عضو جمعیت بر عهده یک نخ قرار داده می شود، که در این صورت زمان اجرای الگوریتم، ضریبی از iteration خواهد شد، به عبارت دیگر ما شبه کد زیر را iteration بار تکرار میکنیم.

شبه کد کرنل بروز رسانی موقعیت هر عضو جمعیت:

شاخص سراسری نخ ها = i

بروز رسانی سطر i ام آرایه d_Values و ذخیره آن در آرایه temp

محاسبه شایستگی عناصر آرایه temp

اگر مقدار شایستگی جدید از مقدار شایستگی d_ Fitness[i] بهینه تر بود، عناصر آرایه temp را در سطر i ام آرایه d_Values قرار می دهیم

۴-۴- مراحل روش پیشنهادی

موازی‌سازی الگوریتم گرده‌افشانی گل در واقع تغییر در ساختار الگوریتم سریال و اضافه نمودن قابلیت موازی جستجو در اعضای جمعیت آن است. در این قسمت مراحل توضیح داده شده در مراحل قبلی را به شکل خلاصه‌تر بیان می‌کنیم:

۱- در ابتدا هر راه‌حل مسئله را در قالب یک گل مدلسازی می‌نمایم که D بعد دارد، برای راحتی در امر موازی سازی مقادیر شایستگی اعضا را در یک آرایه N بعدی قرار می دهیم.

۲- اعضای جمعیت اولیه به شکل تصادفی درون حافظه اصلی میزبان مقداردهی اولیه می‌شوند و یک نسخه از این جمعیت اولیه درون حافظه سراسری دستگاه کپی می‌شود. این مجموعه جمعیت اولیه به شکل یک اشاره‌گر به آرایه $n \times D$ تعریف می‌شود.

۳- هر سطر از آرایه ایجاد شده در حافظه سراسری در واقع یک گل را نشان می‌دهد که توسط یک نخ با شماره منحصربفرد(کودا به هر نخ یک شماره منحصربفرد اختصاص می‌دهد) درون یکی از هسته‌های پردازنده گرافیکی به اجراء گذاشته می‌شود.

۴- هر نخ یک گل را در فضای مسئله مدیریت می‌نماید و یک نسخه از الگوریتم گردافشانی گل‌ها را بر روی داده خود(گل) و درون یک هسته پردازشی کارت گرافیک به اجراء می‌گذارد. با اجرای کرنل موقعیت هر گل به شکل موازی و مستقل در فضای مسئله تعیین می‌شود. در این مرحله بهترین و شایسته‌ترین گل را بهترین گل در این تکرار در نظر می‌گیریم و بررسی می‌کنیم آیا این گل،از بهترین گل تکرار قبلی شایستگی بیشتری دارد یا خیر.

۵- اجرای کرنل به تعداد حداکثر تکرار مجاز در برنامه انجام شده تا گل‌ها به جوابهای مسئله همگرا شوند و در ادامه بهترین گل سراسری که بهترین جواب مسئله است به عنوان بهترین راه‌حل انتخاب می‌شود.

۴-۵- بار محاسباتی الگوریتم

برای محاسبه بار محاسباتی، می توان پیچیدگی کندترین حلقه یا بخش الگوریتم را در نظر گرفت مهمترین قسمت الگوریتم موازی گرده افشانی گل، "کرنل بروز رسانی موقعیت هر عضو جمعیت" می باشد که دارای پیچیدگی محاسباتی $O(n) = dim$ می باشد و از آنجایی که این کرنل به تعداد iteration یا حداکثر تعداد تکرار مجاز الگوریتم اجرا می شود، پیچیدگی برابر می شود با $iteration \times dim$ و در نتیجه بار محاسباتی الگوریتم موازی گرده افشانی گل در بدترین حالت برابر می شود با $W(n) = iteration \times dim$

۴-۶- جمع بندی

معماری و ساختار پردازنده گرافیکی به گونه‌ای است که می‌توان به کمک آن الگوریتم‌های که ماهیتی موازی دارند را به شکل موازی به اجراء گذاشت تا الگوریتم مورد نظر با سرعت و شتاب بیشتری نسبت به حالت سریال اجراء شود. الگوریتم‌های تکاملی ماهیتی موازی دارند و اعضای جمعیت آن به شکل موازی می‌توانند فضای جستجوی مسئله را کاوش نمایند. در این پژوهش الگوریتم تکاملی گرده‌افشانی گل‌ها با استفاده از معماری پردازنده کارت گرافیک و چارچوب کودافای زبان سی‌شارپ توسع و یک نسخه موازی آن ارایه گردید. این فصل الگوریتم پیشنهادی تشریح و مراحل آن بیان شده است در فصل آتی روش پیشنهادی را به کمک تعدادی از توابع ارزیابی ریاضی مورد ارزیابی قرار داده و شتاب الگوریتم پیشنهادی با روش استاندارد نیز مقایسه می‌شود.

فصل ۵- پیاده سازی و تحلیل یافته ها

۵-۱- مقدمه

الگوریتم گرده‌افشانی گل[1] یک الگوریتم تکاملی[2] و الهام گرفته شده بر اساس رفتار زاد و ولد گیاهان است. این الگوریتم با تقلید از فرآیند گرده‌افشانی در گلها سعی می‌کند مسائل بهینه‌سازی[3] سخت و دشوار را حل نماید. در فصل پیشین یک نسخه از الگوریتم گرده‌افشانی گل تحت معماری موازی پردازنده گرافیکی[4] و به کمک چارچوب کودافای[5] ارایه داده شد. در این فصل الگوریتم موازی گرده-افشانی گل با استفاده از توابع ارزیابی ریاضی[6] مورد ارزیابی قرار خواهد گرفت و شتاب الگوریتم پیشنهادی با الگوریتم سریال گرده‌افشانی گل مورد مقایسه قرار خواهد گرفت و در نهایت نسخه پیشنهادی الگوریتم گرده‌افشانی گل‌ها را بر روی کارتهای گرافیک مختلف به اجراء گذاشته می‌شود و تاثیر سخت افزار بر روی شتاب الگوریتم نیز بررسی می‌شود.

[1] flower pollination algorithm
[2] Evolutionary Algorithms
[3] optimization problems
[4] GPU
[5] Cudafy.net
[6] Benchmark function

۵-۲- سیستم و سخت افزار و محیط شبیه سازی

جهت پیاده‌سازی روش پیشنهادی و الگوریتم گرده‌افشانی گل استاندارد از سیستم کامپیوتری با مشخصات سخت‌افزاری مندرج در جدول(۵-۱) استفاده شده است. شتاب و زمان اجرای الگوریتم‌های تکاملی و الگوریتم پیشنهادی گردافشانی گل به مشخصات سخت افزاری سیستم بستگی دارد.

جدول ۵-۱: مشخصات سخت افزار

دستگاه	CPU	GPU
پردازنده	Intel® Core™ i7-3537U	GeForce GT 630M
تعداد هسته	۴ هسته	۹۶ هسته
سرعت پردازش	2.00 Ghz	800 MHz
نوع حافظه	DDR3	GDDR5
اندازه حافظه	8 GB	2 Ghz
سیستم عامل	Windows 10 (64 Bit)	-
ظرفیت محاسباتی	-	2.1
نسخه کودا	-	7.5

پیاده‌سازی الگوریتم پیشنهادی به کمک زبان C++ و چارچوب کودا صورت گرفته است.

۵-۳- ارزیابی روش پیشنهادی

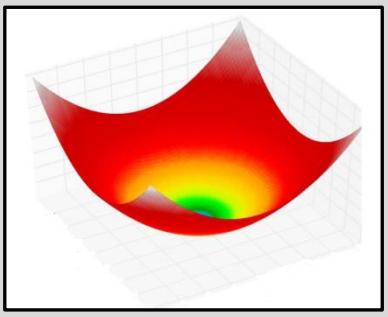
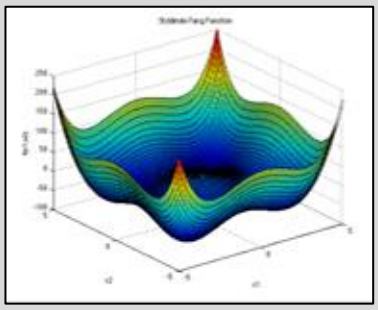
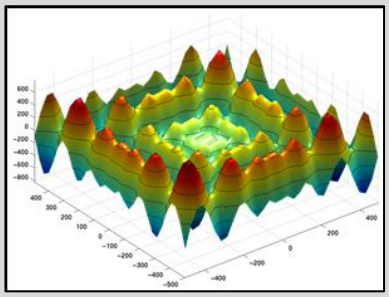
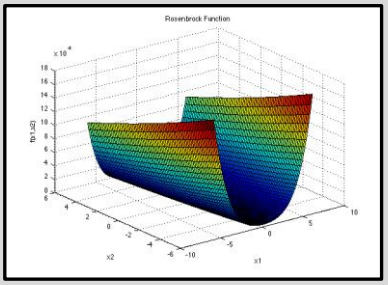
یکی از روش‌های مهم و متداول در ارزیابی الگوریتم موازی‌سازی به وسیله چارچوب کودا یا کودافای استفاده از توابع ارزیابی ریاضی است که نمونه‌های از آن در منابع[۴۰و۳۹و۳۸] ذکر شده است.

۵-۳-۱- توابع ارزیابی

جهت سنجش همگرایی و دقت الگوریتم پیشنهادی با الگوریتم استاندارد، نیاز به یک معیار مشخص و استاندارد تحت عنوان توابع ارزیابی ریاضی است. توابع ارزیابی مختلفی جهت ارزیابی همگرایی و دقت الگوریتم‌های تکاملی بکار گرفته می‌شود که به طور معمول چند مورد از این توابع با پیچیدگی‌های مختلف جهت ارزیابی همگرایی و شتاب الگوریتم‌های تکاملی کافی است. شکل و تعداد ابعاد یک تابع ارزیابی می‌تواند معرف پیچیدگی مسئله مورد نظر باشد. از جمله توابع ارزیابی که در این پژوهش برای

ارزیابی مورد استفاده قرار می‌گیرد توابع موجود در منبع[۳] می‌باشند. در جدول(۵-۲)، چهار تابع ارزیابی بکار رفته در این پژوهش نشان داده شده است:

جدول ۵-۲: توابع ارزیابی بکار رفته جهت محاسبه زمان اجراء و شتاب الگوریتم پیشنهادی

نام تابع	رابطه	محدوده	شکل در فضای سه بعدی
Sphere	$\sum_{i=1}^{D} x_i^2$	$(-10,10)^D$	
Styblinski-tang	$\frac{1}{2}\sum_{i=1}^{D}[x_i^4 - 16x_i^2 + 5x]$	$(-5,5)^D$	
Griewank	$\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}}) + 1$	$(-10,10)^D$	
Rosenbrock	$\sum_{i=1}^{D-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$(-10,10)^D$	

الگوریتم موازی و سریال گرده‌افشانی گل فقط در روند اجراء و زمان اجراء با هم تفاوت دارند که برای بررسی‌ها از معیار رابطه(۵-۱) جهت ارزیابی سرعت و عملکرد الگوریتم پیشنهادی استفاده می‌شود:

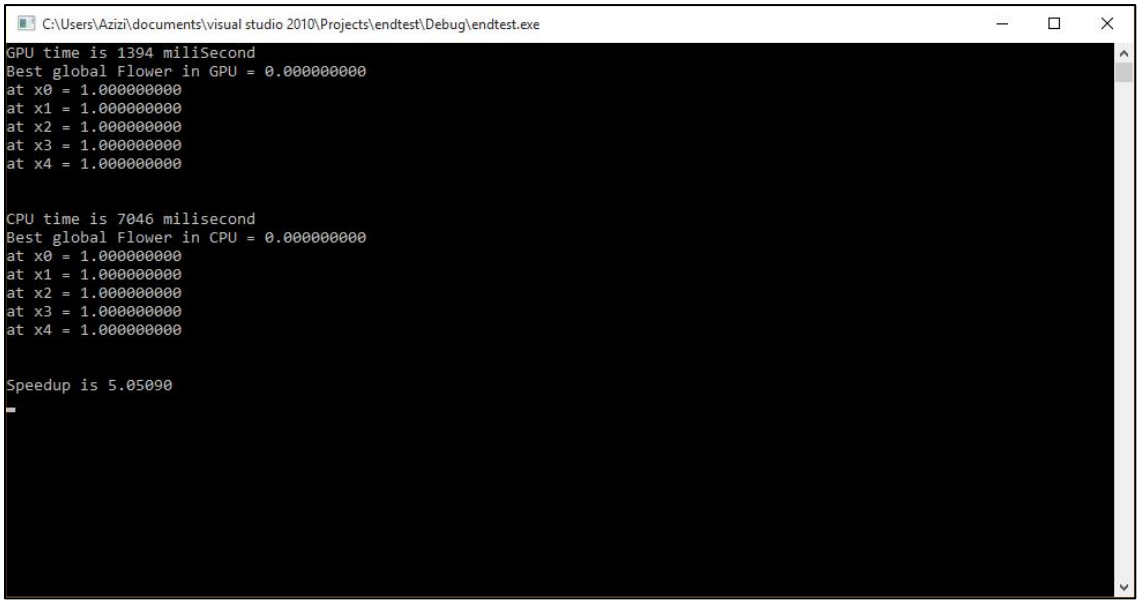
رابطه ۵-۱

$$speedup = \frac{time\ execute\ in\ cpu}{time\ execute\ in\ gpu}$$

در این رابطه شتاب یا سرعت الگوریتم پیشنهادی برابر زمان اجرای الگوریتم در پردازنده اصلی به پردازنده گرافیکی است. ارزیابی‌های این قسمت بر روی سیستم شماره یک انجام گرفته است که مسلماً این زمان و شتاب‌ها به سخت افزاری که برنامه روی آن اجراء می‌شود بستگی دارد. در این قسمت روش پیشنهادی این پایان‌نامه بر روی تعدادی از توابع ارزیابی بکار رفته در منبع همراه با خروجی‌ها نشان داده می‌شود.

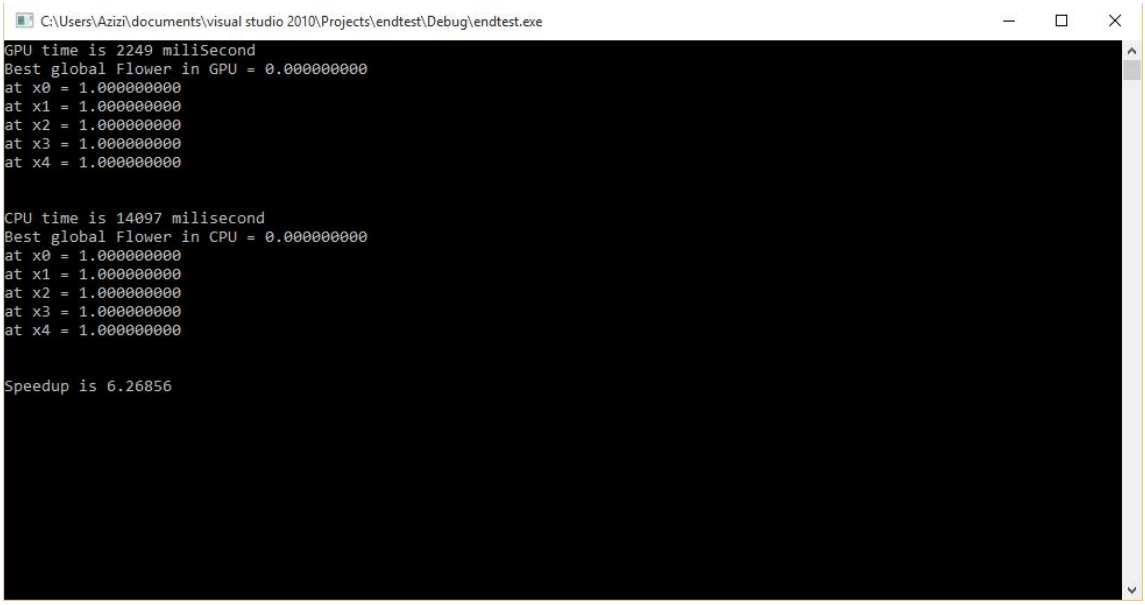
۵-۳-۱-۱- تابع ارزیابی Rosenbrock

تابع ارزیابی Rosenbrock دارای بهینه محلی در نقطه $x_i = 1$ است که مقدار بهینه در این نقطه f^*=0 می‌باشد. در شکل(۵-۱)، خروجی الگوریتم پیشنهادی به همراه مقایسه زمانی حالت اجراء در پردازنده گرافیکی و پردازنده اصلی به ازای جمعیت اولیه ۲۰۰۰، تعداد تکرار ۵۰۰۰ و ابعاد ۵ برای این تابع ارزیابی نمایش داده شده است. در این شکل شتاب الگوریتم پیشنهادی حدوداً ۵.۰۵ محاسبه شده است.



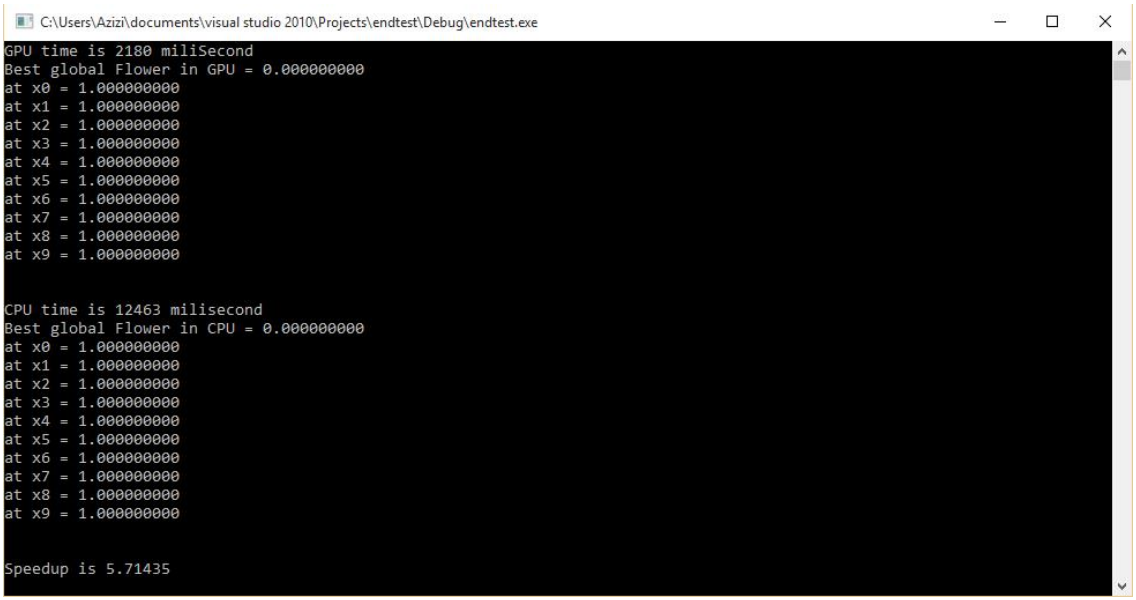
شکل ۵-۱ : خروجی الگوریتم پیشنهادی با جمعیت اولیه ۲۰۰۰ ،تکرار ۵۰۰۰ و ابعاد ۵ (تابع ارزیابی Rosenbrock)

در شکل(۵-۲)، تعداد جمعیت اولیه دو برابر مقدار قبلی یعنی ۴۰۰۰ شده و تعداد تکرار . ابعاد مسئله همان ۵۰۰۰ و ۵ می‌باشد. خروجی الگوریتم پیشنهادی به همراه مقایسه زمانی حالت اجراء در پردازنده گرافیکی و پردازنده اصلی در این حالت نمایش داده شده است. همانگونه که در خروجی برنامه مشخص می‌باشد با افزایش دو برابری جمعیت اولیه، شتاب الگوریتم پیشنهادی نسبت به حالت سریال تقریباً ۶.۲۶ و نسبت به شتاب قبلی ۱.۲۴ برابر شده است.



شکل ۵-۲ : خروجی الگوریتم پیشنهادی با جمعیت اولیه ۴۰۰۰ ،تکرار ۵۰۰۰ و ابعاد ۵ (تابع ارزیابی Rosenbrock)

در شکل(۵-۳)، تعداد جمعیت اولیه و تعداد تکرار نسبت به آزمایش اول ثابت و ابعاد مسئله دو برابر مقدار قبلی یعنی ۱۰ شده است. خروجی الگوریتم پیشنهادی به همراه مقایسه زمانی حالت اجراء در پردازنده گرافیکی و پردازنده اصلی در این حالت نمایش داده شده است. همانگونه که در خروجی برنامه مشخص می‌باشد با افزایش دو برابری ابعاد مسئله، شتاب الگوریتم پیشنهادی نسبت به حالت سریال تقریباً ۵.۷۱ و نسبت به شتاب آزمایش اول ۱.۱۳ برابر شده است.

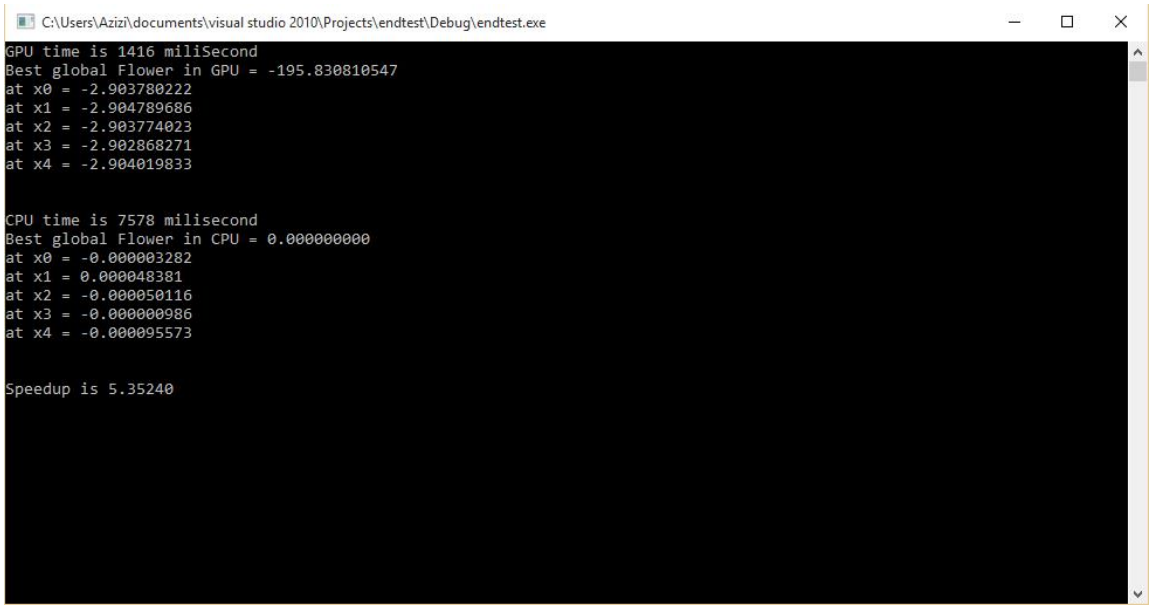


شکل ۵-۳ : خروجی الگوریتم پیشنهادی با جمعیت اولیه ۲۰۰۰ ،تکرار ۵۰۰۰ و ابعاد ۱۰ (تابع ارزیابی Rosenbrock)

این آزمایش و آزمایشات مشابه نشان می‌دهد که افزایش جمعیت اولیه یا افزایش ابعاد مسئله باعث می‌شود که زمان اجراء برنامه در پردازنده اصلی و گرافیکی افزایش یابد به گونه‌ای که این افزایش در پردازنده گرافیکی نسبت به پردازنده اصلی کمتر است. به طور کلی افزایش جمعت اولیه و ابعاد مسئله باعث می‌شود شتاب الگوریتم پیشنهادی افزایش یابد.

۵-۳-۱-۲- تابع ارزیابی Styblinski-tang

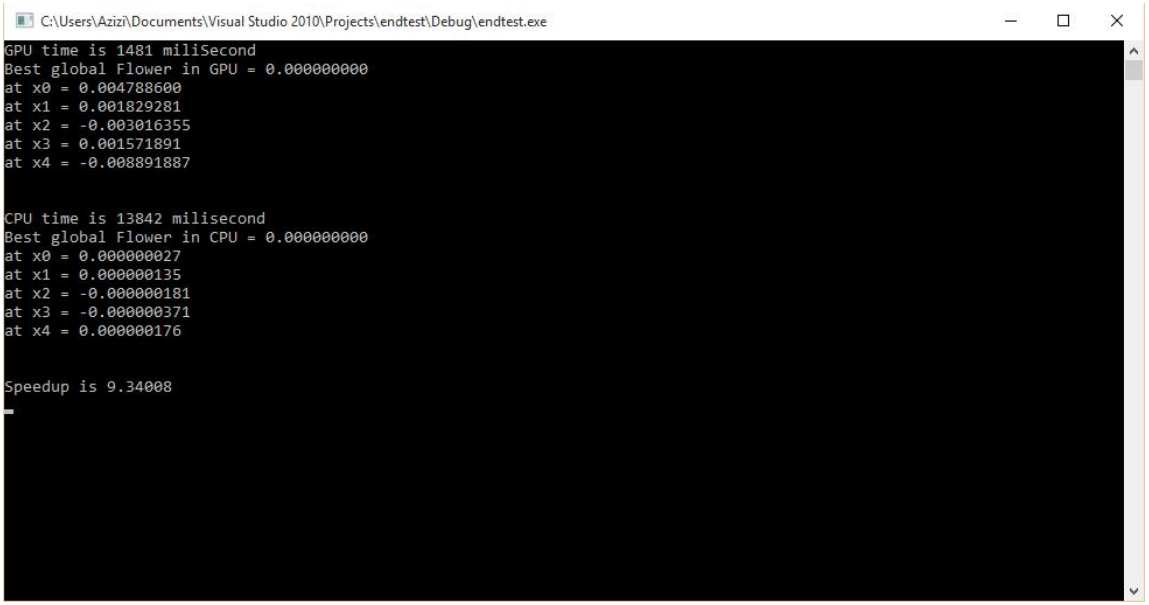
تابع ارزیابی Styblinski-tang دارای بهینه محلی در نقطه $x_i = -2.903534$ است که مقدار بهینه در این نقطه $f^* = -39.16599d$ می‌باشد. در شکل(۵-۴)، خروجی الگوریتم پیشنهادی به همراه مقایسه زمانی حالت اجراء در پردازنده گرافیکی و پردازنده اصلی به ازای جمعیت اولیه ۲۰۰۰ و تعداد تکرار ۵۰۰۰ و ابعاد ۵، نمایش داده شده است. همانگونه که در خروجی برنامه مشخص می‌باشد در این حالت شتاب الگوریتم پیشنهادی نسبت به حالت سریال تقریباً ۵.۳۵ برابر شده است.



شکل ۵-۴ : خروجی الگوریتم پیشنهادی با جمعیت اولیه ۲۰۰۰، تکرار ۵۰۰۰ و ابعاد ۵ (تابع ارزیابی Styblinski-tang)

۵-۳-۱-۳- تابع ارزیابی Griewangk

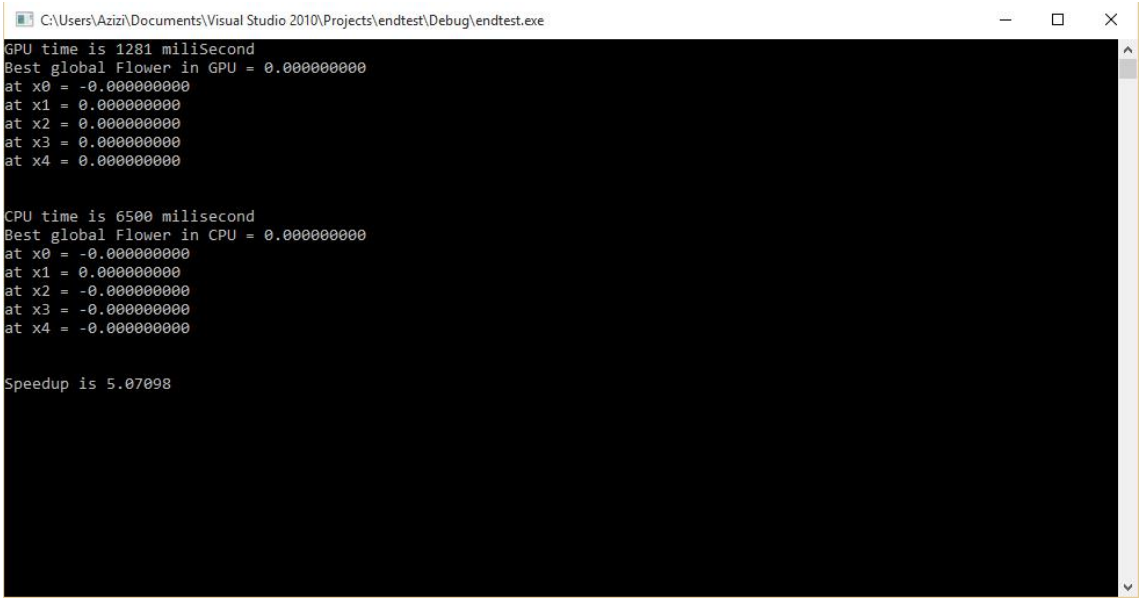
تابع ارزیابی Griewangk دارای بهینه محلی در نقطه$x_i = 0$ است که مقدار بهینه در این نقطه f^*=0 می‌باشد. در شکل(۵-۵)، خروجی الگوریتم پیشنهادی به ازای جمعیت اولیه ۲۰۰۰ ،تعداد تکرار ۵۰۰۰ و ابعاد ۵ نمایش داده شده است. در این حالت شتاب الگوریتم پیشنهادی حدوداً ۹.۳۴ محاسبه شده است.



شکل ۵-۵ : خروجی الگوریتم پیشنهادی با جمعیت اولیه ۲۰۰۰، تکرار ۵۰۰۰ و ابعاد ۵ (تابع ارزیابی Griewangk)

۵-۳-۱-۴- تابع ارزیابی Sphere

تابع ارزیابی Sphere دارای یک بهینه محلی در نقطه $x_i = 0$ است که مقدار بهینه در این نقطه f^*=0 می‌باشد. در شکل(۵-۶)، خروجی الگوریتم پیشنهادی به همراه مقایسه زمانی حالت اجراء در پردازنده گرافیکی و پردازنده اصلی به ازای جمعیت اولیه ۲۰۰۰، تعداد تکرار ۵۰۰۰ و ابعاد مسئله ۵ نمایش داده شده است. همانگونه که در خروجی برنامه مشخص می‌باشد در این حالت شتاب الگوریتم پیشنهادی نسبت به حالت سریال تقریباً ۴.۷۰ برابر شده است.



شکل ۵-۶ : خروجی الگوریتم پیشنهادی با جمعیت اولیه ۲۰۰۰، تکرار ۵۰۰۰ و ابعاد ۵ (تابع ارزیابی Sphere)

۵-۴- شتاب الگوریتم پیشنهادی

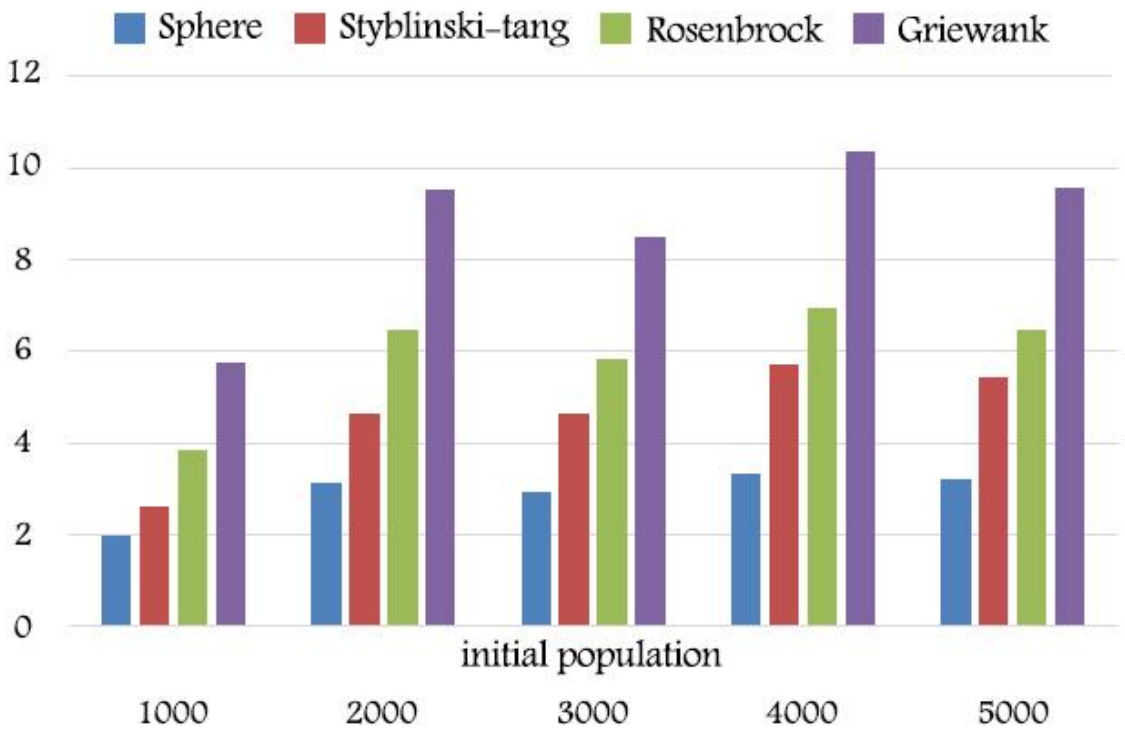
یکی از معیارهای ارزیابی الگوریتم‌های موازی شده با چارچوب کودا و کودافای، محاسبه میانگین شتاب الگوریتم موازی است. جهت محاسبه میانگین شتاب کافی است آزمایش را به تعداد مشخص و بر روی توابع ارزیابی مختلف انجام دهیم و مجموع شتاب الگوریتم پیشنهادی را بر تعداد آزمایشات تقسیم نمایم. نحوه محاسبه میانگین شتاب در رابطه(۵-۲) نشان داده شده است.

رابطه ۵-۲

$$\overline{speedup} = \frac{\sum_{i=1}^{n} |speedup_i|}{n} = \frac{|speedup_1| + |speedup_2| + \cdots + |speedup_n|}{n}$$

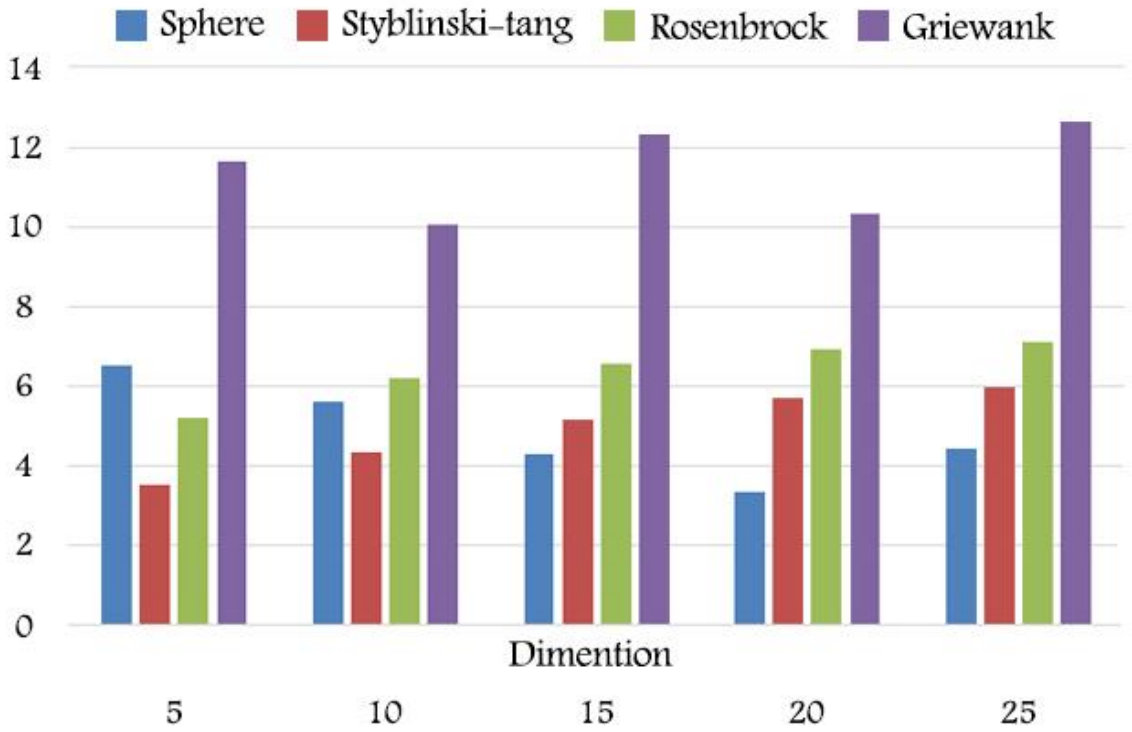
در رابطه بالا، $speedup_i$ و n به ترتیب شتاب هر آزمایش و تعداد آزمایش انجام گرفته شده می‌باشد. و ما در دو حالت میانگین شتاب را محاسبه کردیم:

۱- میزان شتاب متوسط الگوریتم پیشنهادی بر روی چهار تابع ارزیابی جدول(۵-۲) با تعداد ۵۰ آزمایش ، ابعاد مسئله ثابت ۲۰ و تعداد جمعیت اولیه متغیر ۱۰۰۰، ۲۰۰۰، ۳۰۰۰ ، ۴۰۰۰ و ۵۰۰۰ در سیستم جدول (۵-۲) محاسبه و در شکل (۵-۷) به تصویر کشیده شده است.



شکل ۵-۷ : شتاب الگوریتم پیشنهادی در چهار تابع ارزیابی و پیاده سازی شده با جمعیت اولیه متغیر

۲- میزان شتاب متوسط الگوریتم پیشنهادی بر روی چهار تابع ارزیابی جدول(۵-۲) با تعداد ۵۰ آزمایش ، تعداد جمعیت اولیه ثابت ۴۰۰۰ و ابعاد مسئله متغیر ۵، ۱۰، ۱۵ ، ۲۰ و ۲۵ در سیستم جدول (۵-۲) محاسبه و در شکل (۵-۸) به تصویر کشیده شده است



شکل ۵-۸ : شتاب الگوریتم پیشنهادی در چهار تابع ارزیابی و پیاده سازی شده با ابعاد متغیر

تحلیل دو نمودار نشان می‌دهد که با افزایش اندازه جمعیت اولیه میزان شتاب در الگوریتم گرده‌افشانی گل پیشنهادی افزایش می‌یابد در حالی که با افزایش ابعاد مسئله، شتاب الگوریتم به صورت غیر خطی دچار نوسان می شود. این نتایج نشان می‌دهد که موازی‌سازی کودا زمانی مفید است که داده‌های مسئله بقدر کافی زیاد باشند. کاهش اندازه جمعیت اولیه باعث می‌شود که داده‌های اندکی از حافظه میزبان به حافظه سراسری منتقل شوند و این حالت در معماری کودا بهینه نیست زیرا کودا زمانی بهره‌وری را تضمین می‌کند که داده‌های زیای برای پردازش داشته باشد. و از آنجایی که در الگوریتم پیشنهادی همه ابعاد توسط یک نخ مدیریت می شود و سرعت هر نخ از سرعت CPU کمتر می باشد، انتظار داریم با افزایش ابعاد مسئله شتاب الگوریتم کاهش یابد، اما به دلیل مدیریت داده های مسئله توسط ساختار موازی کودا، با افزایش ابعاد مسئله، شتاب الگوریتم دچار نوسان می شود.

نکته دیگری که در نمودارها قابل مشاهده است رابطه شتاب با پیچیدگی تابع ارزیابی ریاضی است که هر چقدر میزان پیچیدگی این توابع بیشتر باشد شتاب الگوریتم پیشنهادی نیز افزایش می یابد.

۵-۵- جمع بندی

در این فصل نسخه الگوریتم پیشنهادی و موازی گرده‌افشانی گل به کمک برنامه‌نویسی کودا در محیط C++ پیاده‌سازی شد و آزمایشات مختلفی به کمک توابع ارزیابی ریاضی جهت محاسبه شتاب و زمان اجرای الگوریتم پیشنهادی و الگوریتم استاندارد گرده‌افشانی گل انجام شد که در این فصل تعدادی از این آزمایشات تشریح و نتایج میانگین شتاب الگوریتم پیشنهادی نیز تا حدودی ارایه شد.

فصل ۶- نتیجه گیری و پیشنهادات آتی

۶-۱- مقدمه

تکامل به ما آموخته است که جاندارانی که بتوانند با گذر زمان با تغییرات محیطی تطبیق بیشتری پیدا نمایند شانس بقای بیشتری خواهند داشت. گیاهان یکی از موجودات زنده‌ای است که طی میلیون‌ها سال توانسته‌اند به بقای خود ادامه دهند و برای افزایش شانس بقاء اندامی مانند گل برای تولید مثل در آنها به وجود آمده است. گل یک اندام اختصاصی و جنسی در گیاهان پیشرفته است که در گیاهان ابتدایی مانند سرخس‌ها وجود نداشته و با گذر زمان و با انتخاب طبیعی[۱] این عضو در آنها به وجود آمده است. گل‌ها نقش مهمی در چرخه تولید مثل گیاهان گلدار ایفاء می‌نماید و گل‌های که موفق به گرده‌افشانی می‌شوند در نهایت عمر بیشتری می‌نمایند تا در نهایت به میوه و دانه تبدیل می-شوند. الگوریتم‌های تکاملی[۲] دسته‌ای از الگوریتم‌های جستجوی هوشمند در فضای جستجوی مسئله می‌باشند که بر اساس فرآیندهای طبیعی الهام گرفته شده است. الگوریتم گرده‌افشانی گل[۳] یک الگوریتم تکاملی مبتنی بر رفتار زاد و ولد گیاهان به کمک گرده‌افشانی آنها می‌باشد. الگوریتم‌های تکاملی روشی مناسب برای حل مسائل بهینه‌سازی[۴] است که می‌توانند جوابهای قابل قبولی را در زمان مناسب ارایه نمایند. الگوریتم‌های تکاملی نظیر گرده‌افشانی گل با افزایش جمعیت اولیه و ابعاد مسئله زمان اجرای آنها افزایش می‌یابد. یکی از ویژگی‌های الگوریتم‌های تکاملی، ماهیت اجرای موازی اعضای جمعیت اولیه در فضای مسئله است که این دسته از الگوریتم‌ها را می‌توان به شیوه‌های موازی‌سازی به اجراء گذاشته شود. در این شیوه هر کدام از اعضای جمعیت به شکل موازی فضای مسئله را مورد جستجو قرار داده که می‌تواند زمان اجراء و رسیدن به بهینه سراسری را کاهش دهد. تاکنون شیوه-های مختلفی جهت موازی سازی الگوریتم‌ها ارایه شده است که برنامه‌نویسی تحت پردازنده گرافیکی[۵] و با چارچوب کودا[۶] یکی از شیوه‌های نوین و جدید موازی سازی محسوب می‌شود. در پردازنده‌های گرافیکی هسته‌های محاسباتی بسیار بیشتری نسبت به پردازنده اصلی[۷] وجود دارد که قادرند که عملیات‌های ساده را به تعداد بسیار زیاد انجام دهند. در این پژوهش هر گل به عنوان عضو جمعیت الگوریتم گرده‌افشانی گل‌ها به شکل مستقل و موازی درون یک هسته[۸] پردازشی پردازنده گرافیکی به اجراء گذاشته مشده است. نتایج پژوهش نشان می‌دهد که روش پیشنهادی دارای سرعت اجرای بالاتری نسبت به الگوریتم استاندارد گرده‌افشانی گل می‌باشد. در این پژوهش آزمایشات مختلفی جهت ارزیابی روش پیشنهادی صورت گرفته است که نتایج حاصل از آن در این فصل ارایه شده است و در انتهای فصل پیشنهادات آتی جهت بهبود یا گسترش روش پیشنهادی بیان شده است.

[1] natural selection
[2] Evolutionary algorithms
[3] flower pollination algorithm
[4] Optimization problems
[5] Graphics Processing Unit (GPU)
[6] Compute Unified Device Architecture(CUDA)
[7] CPU
[8] Core

۶-۲- نتایج کلی

نتایج این پژوهش به شرح زیر خلاصه شده است:

- افزایش تعداد اعضای جمعیت اولیه باعث می‌شود زمان اجرای الگوریتم موازی گرده‌افشانی گل و الگوریتم غیرموازی گرده‌افشانی، افزایش یابد و این افزایش در الگوریتم غیرموازی بیشتر از الگوریتم موازی است.
- افزایش تعداد اعضای جمعیت اولیه باعث می‌شود شتاب الگوریتم پیشنهادی افزایش یابد به عبارتی ساده‌تر برنامه نویسی موازی با کودا زمانی کارایی مناسبی دارد که داده‌ها و ورودی مسئله به اندازه کافی بزرگ باشد.
- شتاب الگوریتم پیشنهادی با پیچیدگی مسئله ارتباطی نزدیک دارد بطوریکه هر چقدر میزان پیچیدگی مسئله بیشتر باشد شتاب الگوریتم پیشنهادی نسبت به مسائل ساده‌تر کاهش می-یابد. به عبارت بهتر شتاب الگوریتم پیشنهادی با افزایش پیچیدگی مسئله اندکی کاهش می-یابد.
- با افزایش ابعاد مسئله، به دلیل سرعت پایین تر هر نخ نسبت به CPU انتظار کاهش شتاب را داریم ، ولی به دلیل استفاده از ساختار موازی کودا در پردازش داده های بزرگ مسئله، در نهایت با افزایش ابعاد مسئله شتاب الگوریتم دچار نوسان می شود.

۶-۳- پیشنهادات آتی

در پژوهش‌های آتی قصد داریم نسخه الگوریتم پیشنهادی را به کمک چند کارت گرافیک پیاده‌سازی نمایم تا شتاب و سرعت الگوریتم پیشنهادی افزایش یابد. یکی دیگر از پژوهش‌های آتی ما بکارگیری الگوریتم پیشنهادی در کاربردهای روزمره و عملی است، همچنین در الگوریتم پیشنهادی هر یک از اعضای جمعیت (گل) توسط یک نخ مدیریت می شود، در نتیجه هر نخ به مدیریت D بعد (D خانه) می پردازد، درنتیجه می توان راهکاری در نظر گرفت که هر یک از ابعاد هر گل هم توسط یک نخ مدیریت شود که در نتیجه شتاب الگوریتم به شدت افزایش خواهد یافت.

یکی از چالش های برخی الگوریتم های تکاملی و الگوریتم گرده افشانی گل تولید اعداد تصادفی بدون تکرار است . طراحی یک الگوریتم تولید عدد تصادفی بدون تکرار و البته سریع می تواند به افزایش شتاب الگوریتم کمک زیادی کند.

۶-۴- جمع بندی و نتیجه گیری

در این پژوهش به کمک معماری موازی پردازنده گرافیکی و با استفاده از امکانات چارچوب کودا، الگوریتم گرده‌افشانی گل موازی شده و نسخه پیشنهادی از نظر سرعت و شتاب با الگوریتم استاندارد

گرده‌افشانی گل مورد مقایسه قرار گرفته شد. نتایج کلی نشان می‌دهد که زمان اجراء و سرعت الگوریتم پیشنهادی بویژه زمانیکه اندازه داده‌ها بقدر کافی بزرگ باشد نسبت به الگوریتم استاندارد گرده‌افشانی گل بیشتر است.

منابع:

1. Yang, X. S. (2010). Engineering optimization: an introduction with metaheuristic applications. John Wiley & Sons.
2. Dasgupta, D., & Michalewicz, Z. (Eds.). (2013). *Evolutionary algorithms in engineering applications*. Springer Science & Business Media.
3. Binitha, S., & Sathya, S. S. (2012). A survey of bio inspired optimization algorithms. *International Journal of Soft Computing and Engineering*, *2*(2), 137-151.
4. Alatas, B. (2011). Photosynthetic algorithm approaches for bioinformatics.*Expert Systems with Applications*, *38*(8), 10541-10546.
5. Mehrabian, A. R., & Lucas, C. (2006). A novel numerical optimization algorithm inspired from weed colonization. *Ecological informatics*, *1*(4), 355-366.
6. Yang, X. S. (2012). Flower pollination algorithm for global optimization. In*Unconventional computation and natural computation* (pp. 240-249). Springer Berlin Heidelberg.
7. Nvidia, C. U. D. A. (2008). Programming guide.
8. Yang, X. S., & He, X. (2013). Firefly algorithm: recent advances and applications. *International Journal of Swarm Intelligence*, *1*(1), 36-50.
9. Tariq, S. (2011). An Introduction to GPU Computing and CUDA Architecture.*NVIDIA Corporation*.
10. Cheng, J., Grossman, M., & McKercher, T. (2014). *Professional Cuda C Programming*. John Wiley & Sons.
11. Ahrari, A., Saadatmand, M. R., Shariat-Panahi, M., & Atai, A. A. (2010). On the limitations of classical benchmark functions for evaluating robustness of evolutionary algorithms. *Applied Mathematics and Computation*, *215*(9), 3222-3229.
12. http://biology.tutorvista.com/plant-kingdom/types-of-pollination.html
13. Yang, X. S. (2010). Firefly algorithm, Levy flights and global optimization. In*Research and development in intelligent systems XXVI* (pp. 209-218). Springer London.
14. Shah-Hosseini, H. (2009). The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm. *International Journal of Bio-Inspired Computation*, *1*(1-2), 71-79.
15. Eskandar, H., Sadollah, A., Bahreininejad, A., & Hamdi, M. (2012). Water cycle algorithm–A novel metaheuristic optimization method for solving constrained engineering optimization problems. *Computers & Structures*, *110*, 151-166.
16. andomi, A. H., & Alavi, A. H. (2012). Krill herd: a new bio-inspired optimization algorithm. *Communications in Nonlinear Science and Numerical Simulation*, *17*(12), 4831-4845.
17. Mussi, L., Daolio, F., & Cagnoni, S. (2011). Evaluation of parallel particle swarm optimization algorithms within the CUDA™ architecture. *Information Sciences*, *181*(20), 4642-4657.

18. Husselmann, A. V., & Hawick, K. A. (2012, March). Parallel parametric optimisation with firefly algorithms on graphical processing units. In *Proc. Int. Conf. on Genetic and Evolutionary Methods (GEM'12)* (pp. 77-83).
19. Pospichal, P., Jaros, J., & Schwarz, J. (2010). Parallel genetic algorithm on the cuda architecture. In *Applications of Evolutionary Computation* (pp. 442-451). Springer Berlin Heidelberg.
20. Cheng, M. Y., & Prayogo, D. (2014). Symbiotic Organisms Search: A new metaheuristic optimization algorithm. *Computers & Structures*, *139*, 98-112.
21. Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine learning*, *3*(2), 95-99.
22. Eberhart, R. C., & Shi, Y. (2001). Particle swarm optimization: developments, applications and resources. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on* (Vol. 1, pp. 81-86). IEEE.
23. Yang, X. S. (2010). A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)* (pp. 65-74). Springer Berlin Heidelberg.
24. CUDA, C. (2013). Programming Guide v5. 5. *NVIDIA Corporation, July*.
25. Dotzler, G., Veldema, R., & Klemm, M. (2010, May). Jcudamp: Openmp/java on cuda. In *Proceedings of the 3rd International Workshop on Multicore Software Engineering* (pp. 10-17). ACM.
26. Knas, M., & Cierniak, R. (2014). A Modification of the Parallel Spline Interpolation Algorithms. In *Image Processing and Communications Challenges 5* (pp. 185-190). Springer International Publishing.
27. Kumar, J., Singh, L., & Paul, S. (2013, July). GPU based parallel cooperative particle swarm optimization using C-CUDA: a case study. In *Fuzzy Systems (FUZZ), 2013 IEEE International Conference on* (pp. 1-8). IEEE.
28. www.hybriddsp.com/cudafy/**CUDAfy_User_Manual_1_22.pdf**
29. Li, X. L., Shao, Z. J., & Qian, J. X. (2002). An optimizing method based on autonomous animats: fish-swarm algorithm. *System Engineering Theory and Practice*, *22*(11), 32-38.
30. NVIDIA Corporation: NVIDIA CUDA compute unified device architecture programming guide. http://developer.nvidia.com/object/cuda_3_2_toolkit_rc.html, 2010.
31. Zhou, Y., & Tan, Y. (2009, May). GPU-based parallel particle swarm optimization. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*(pp. 1493-1500). IEEE.
32. de P Veronese, L., & Krohling, R. (2009, May). Swarm's flight: accelerating the particles using C-CUDA. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on* (pp. 3264-3270). IEEE.
33. Bai, H., OuYang, D., Li, X., He, L., & Yu, H. (2009, December). MAX-MIN ant system on GPU with CUDA. In *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on* (pp. 801-804). IEEE.
34. Hu, Y., Yu, B., Ma, J., & Chen, T. (2011, May). Parallel fish swarm algorithm based on GPU-acceleration. In *Intelligent Systems and Applications (ISA), 2011 3rd International Workshop on* (pp. 1-4). IEEE.

35. Luo, G. H., Huang, S. K., Chang, Y. S., & Yuan, S. M. (2014). A parallel Bees Algorithm implementation on GPU. *Journal of Systems Architecture*, *60*(3), 271-279.
36. Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., & Zaidi, M. (2011, July). The Bees Algorithm–A Novel Tool for Complex Optimisation. In*Intelligent Production Machines and Systems-2nd I* PROMS Virtual International Conference 3-14 July 2006* (p. 454). Elsevier.
37. Pham, D. T., & Castellani, M. (2009). The Bees Algorithm: modelling foraging behaviour to solve continuous optimization problems. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, *223*(12), 2919-2938.
38. Kumar, J., Singh, L., & Paul, S. (2013, July). GPU based parallel cooperative particle swarm optimization using C-CUDA: a case study. In *Fuzzy Systems (FUZZ), 2013 IEEE International Conference on* (pp. 1-8). IEEE.
39. Husselmann, A. V., & Hawick, K. A. (2012, March). Parallel parametric optimisation with firefly algorithms on graphical processing units. In *Proc. Int. Conf. on Genetic and Evolutionary Methods (GEM'12)* (pp. 77-83).
40. Lan, T., Guo, M., Qu, J., Chai, Y., Liu, Z., & Zhang, X. (2015, May). CUDA-based hierarchical multi-block particle swarm optimization algorithm. In *Control and Decision Conference (CCDC), 2015 27th Chinese* (pp. 4419-4423). IEEE
41. Rabinovich, M., et al. (2012). Particle swarm optimization on a gpu. Electro/Information Technology (EIT), 2012 IEEE International Conference on, IEEE.
42. Van Luong, T., et al. (2013). "GPU computing for parallel local search metaheuristic algorithms." Computers, IEEE Transactions on 62(1): 173-185.
43. Arora, R., et al. (2010). Parallelization of binary and real-coded genetic algorithms on GPU using CUDA. Evolutionary Computation (CEC), 2010 IEEE Congress on, IEEE.
44. Husselmann, A. V. and K. Hawick (2012). Parallel parametric optimisation with firefly algorithms on graphical processing units. Proc. Int. Conf. on Genetic and Evolutionary Methods (GEM'12).
45. Cardenas-Montes, M., et al. (2011). Accelerating particle swarm algorithm with GPGPU. Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on, IEEE
46. Calazan, R. M., et al. (2013). Three alternatives for parallel GPU-based implementations of high performance particle swarm optimization. Advances in Computational Intelligence, Springer: 241-252.
47. Hung, Y. and W. Wang (2012). "Accelerating parallel particle swarm optimization via GPU." Optimization Methods and Software 27(1): 33-51.
48. Sato, Y., et al. (2011). GPU acceleration for Sudoku solution with genetic operations. Evolutionary Computation (CEC), 2011 IEEE Congress on, IEEE

Abstract

Evolutionary algorithms are a subset of evolutionary computations and include the algorithms the search starts with multiple points in solution space to discover the answer and they are used in problems that ordinary solutions are not responsive due to complexity and various parameters of the problem.
Evolutionary algorithms inspired by nature and the laws governing it try to calculate the optimal solution to difficult problems accurately. So far, various evolutionary algorithms are presented based on the social behavior of organisms, biological behavior and physical phenomena. Pollination of flowers is an interesting process used by many plant species for reproduction and survival, and researchers have used the behavior of the plants to create an evolutionary algorithm called flower pollination algorithm.
Finding the optimal solutions to an optimization problem is challenging and difficult by increasing the size of the problem by evolutionary algorithms such as flower pollination algorithm and the flower pollination algorithm may converge to local optimums instead of a global optimization which is solved by increasing the population and repetition of flower pollination algorithm which increases the running time significantly.
In this study we try to use the members of the initial population parallel in the graphics processing unit which has higher processor cores than the main processor. The results of the tests and simulations indicate that the parallel version of the flower pollination algorithm by the parallelization technology of graphics (CUDA) has higher execution speed than standard flower pollination algorithm.

Keywords:

Evolutionary algorithms, flower pollination algorithm, Graphics Processing Unit, CUDA, a parallel version of flower pollination algorithm

Shahrood University of Technology

Faculty of e-learning

Accelerating flower pollination algorithm using GPU

Morteza Azizi Sales

Supervisor:

Prof. Hamid Hassanpour

February 2016