ارائه یک الگوریتم موازی بهینه‌سازی غذایابی باکتری پیاده‌سازی شده در واحد پردازش گرافیکی

علی رفیعی و سیدمرتضی موسوی

چکیده: الگوریتم غذایابی باکتری یکی از الگوریتم‌های بهینه‌سازی مبتنی بر جمعیت است که برای حل مسایل جستجو در شاخه‌های مختلف علوم استفاده می‌شود. یکی از مواردی که امروزه مورد توجه قرار گرفته است قابلیت اجرای موازی الگوریتم‌های بهینه‌سازی مبتنی بر جمعیت در پردازنده‌های گرافیکی است. با توجه به سرعت پایین الگوریتم بهینه‌سازی غذایابی باکتری در مواجهه با مسایل پیچیده و همچنین عدم توانایی حل مسایل با ابعاد بزرگ توسط این الگوریتم، اجرای آن بر روی پردازنده‌های گرافیکی یک راه حل مناسب برای پوشش نقاط ضعف این الگوریتم می‌باشد. در این نوشته ما یک نسخه موازی از الگوریتم بهینه‌سازی غذایابی باکتری ارائه دادیم که قابلیت اجرا در پردازنده‌های گرافیکی و با استفاده از طراحی کودا را دارد. همچنین کارایی این الگوریتم را با استفاده از تعدادی از مسایل شناخته‌شده بهینه‌سازی در مقایسه با الگوریتم استاندارد بهینه‌سازی غذایابی باکتری مورد ارزیابی قرار دادیم. نتایج نشان می‌دهد که الگوریتم موازی غذایابی باکتری نسبت به الگوریتم استاندارد غذایابی باکتری دارای سرعت و کارایی بالاتری می‌باشد.

کلیدواژه: **الگوریتم مبتنی بر جمعیت، الگوریتم موازی غذایابی باکتری، کودا، واحد پردازش گرافیکی.**

۱- مقدمه

الگوریتم‌های بهینه‌سازی اهمیت زیادی در بسیاری از شاخه‌های علوم دارند. به عنوان مثال فیزیکدان‌ها، شیمیدان‌ها و مهندسان علاقه دارند تا یک طرح بهینه برای طراحی یک پروسه شیمیایی به کار برند و یا محصول تولیدشده را با داشتن شروطی مثل هزینه و آلودگی کم، بیشینه کنند. همچنین در برازش غیر خطی مدل و منحنی نیز به نوعی به بهینه‌سازی نیاز داریم. اقتصاددانان و تحقیق‌کنندگان در عملیات نیز باید جایابی بهینه منابع در جامعه و صنعت را پیدا کنند. الگوریتم‌های بهینه‌سازی به دو دسته الگوریتم‌های دقیق و الگوریتم‌های تقریبی تقسیم‌بندی می‌شوند. الگوریتم‌های دقیق قادر به یافتن جواب بهینه به صورت دقیق هستند اما در مورد مسایل بهینه‌سازی سخت کارایی ندارند و زمان حل آنها در این مسایل به صورت نمایی افزایش می‌یابد. الگوریتم‌های تقریبی قادر به یافتن جواب‌های خوب (نزدیک به بهینه) در زمان حل کوتاه برای مسایل بهینه‌سازی سخت هستند. الگوریتم‌های تقریبی نیز به دو دسته الگوریتم‌های ابتکاری و فراابتکاری بخش‌بندی می‌شوند. مشکل اصلی الگوریتم‌های ابتکاری، قرارگرفتن آنها در نقاط بهینه محلی و ناتوانی آنها برای کاربرد در مسایل گوناگون است.

این مقاله در تاریخ ۱۸ اردیبهشت ماه ۱۳۹۵ دریافت و در تاریخ ۱۲ فروردین ماه ۱۳۹۶ بازنگری شد.

علی رفیعی، گروه کامپیوتر، دانشکده فنی و مهندسی، دانشگاه آزاد اسلامی واحد اراک، اراک، (email: rafiee.au@gmail.com).

سیدمرتضی موسوی، گروه کامپیوتر، دانشکده فنی و مهندسی، دانشگاه آزاد اسلامی واحد اراک، اراک، (email: m_mosavi@iau-arak.ac.ir).

الگوریتم‌های فراابتکاری برای حل این مشکلات الگوریتم‌های ابتکاری ارائه شده‌اند. در واقع الگوریتم‌های فراابتکاری، یکی از انواع الگوریتم‌های بهینه‌سازی تقریبی هستند که دارای راهکارهای برون‌رفت از نقاط بهینه محلی می‌باشند و قابل کاربرد در طیف گسترده‌ای از مسایل هستند. رده‌های گوناگونی از این نوع الگوریتم در دهه‌های اخیر توسعه یافته که از مهم‌ترین آنها می‌توان به الگوریتم‌های فراابتکاری مبتنی بر جمعیت اشاره کرد. تعدادی از معروف‌ترین این الگوریتم‌ها، الگوریتم‌های ژنتیک [۱]، الگوریتم بهینه‌سازی ذرات [۲]، الگوریتم مورچه [۳] و الگوریتم زنبور عسل [۴] می‌باشند.

یکی دیگر از الگوریتم‌های مبتنی بر جمعیت که در شاخه‌های مختلف علوم مورد استفاده قرار گرفته است الگوریتم بهینه‌سازی غذایابی باکتری می‌باشد که از رفتار طبیعی باکتری‌ها برای یافتن منابع غذایی الهام گرفته است.

بالابردن سرعت و کارایی این الگوریتم‌ها همواره از مسایل مورد نظر محققان بوده و برای این منظور از رهیافت‌های مختلفی استفاده شده است. ترکیب این الگوریتم‌ها با یکدیگر و ساخت الگوریتم‌های هیبریدی یکی از روش‌های بالابردن کارایی الگوریتم‌های مبتنی بر جمعیت است که در آن نقاط ضعف یک الگوریتم با مدل‌های قدرتمند موجود در الگوریتم‌های دیگر جایگزین می‌شود. همچنین اجرای موازی این الگوریتم‌ها یکی از رهیافت‌های بالابردن سرعت اجرای آنها می‌باشد. با توجه به وجود ذرات مصنوعی در تمامی این الگوریتم‌ها، پتانسیل خوبی برای موازی‌سازی آنها وجود دارد. اجرای موازی این الگوریتم‌ها در محیط‌های سخت‌افزاری مختلفی صورت گرفته است. پردازنده‌های گرافیکی از محیط‌های پردازش موازی بسیار سریع می‌باشند که امروزه مورد توجه محققان برای اجرای برنامه‌های موازی قرار گرفته است. همچنین برای ارتباط با سخت‌افزار پردازنده‌های گرافیکی و آسان‌نمودن برنامه‌نویسی برای آنها، بسترهای نرم‌افزاری مختلفی طراحی و ارائه شده است. یکی از قوی‌ترین و محبوب‌ترین آنها معماری کودا می‌باشد که توسط شرکت انویدیا برای برنامه‌نویسی و طراحی کدهای موازی بر روی پردازنده‌های گرافیکی این شرکت طراحی شده است.

الگوریتم‌های مبتنی بر جمعیت زیادی با استفاده از معماری کودا برای اجرای موازی بر روی پردازنده‌های گرافیکی ارائه شده که سرعت و کارایی الگوریتم‌ها را تا ده‌ها برابر افزایش داده است. برای مثال در [۵] یک نسخه الگوریتم ژنتیک به صورت موازی در معماری کودا پیاده‌سازی شده است که در این الگوریتم از روش جزیره استفاده می‌شود و با استفاده از حافظه اشتراکی در پردازنده گرافیکی نیاز به استفاده از گذرگاه داده سیستم را از میان برده است و سرعت اجرا به واسطه بالارفتن سرعت ارتباط، افزایش یافته است. در این الگوریتم برای هر رشته از الگوریتم، یک نخ در پردازنده گرافیکی در نظر گرفته شده است. مهاجرت بین جزیره‌ها توسط حافظه اصلی در پردازنده گرافیکی انجام می‌شود و برای

مرتب‌سازی مقادیر تابع تناسب از الگوریتم مرتب‌سازی بایتونیک استفاده شده است. همچنین در [۶] نویسندگان با استفاده از الگوریتم بهینه‌سازی زنبور و تکنولوژی کودا یک الگوریتم موازی زنبور به نام CUBA طراحی کرده‌اند که قابلیت اجرا بر روی سخت‌افزار موجود در پردازنده گرافیکی را دارد. در این الگوریتم در هر بلوک از سخت‌افزار کارت گرافیکی، چند کلونی زنبور پیاده‌سازی شده است. ارتباط بین کلونی‌ها از طریق حافظه اشتراکی انجام می‌شود که باعث بالارفتن سرعت ارتباط خواهد شد. همچنین طراحان برای بالابردن سرعت اجرا از تکنیک مرتب‌سازی زوج و فرد برای مرتب‌سازی مقادیر تابع هدف زنبورها به صورت موازی استفاده کرده اند که این امر کارایی الگوریتم را بالا برده است. سرعت دسترسی به جواب برای چندین تابع معیار بین ۱۳ تا ۵۶ برابر سریع‌تر از الگوریتم ترتیبی زنبور است. در [۷] نویسندگان یک الگوریتم تراکم ذرات موازی طراحی کرده‌اند که با استفاده از معماری کودا بر روی پردازنده‌های گرافیکی شرکت Nvidia قابل اجرا می‌باشد. در این الگوریتم سازمان داده‌ها در داخل پردازنده گرافیکی به صورت آرایه‌ای یک‌بعدی طراحی شده است. همچنین نکته جالب توجه که سرعت این الگوریتم را افزایش داده است تولید اعداد تصادفی مورد نیاز اجرای الگوریتم در ابتدای عملیات و قبل از شروع اجرای الگوریتم می‌باشد. اجرای این الگوریتم برای چند تابع معیار معروف، افزایش سرعت ۱۱برابری را نشان می‌دهد. محققان در [۸] یک الگوریتم کلونی مورچه موازی قابل اجرا در سخت‌افزار پردازنده گرافیکی ارائه کرده‌اند. این الگوریتم بر اساس الگوریتم استاندارد مورچه ماکسیمم- مینیمم طراحی شده است. از این الگوریتم برای حل مسئله فروشنده دوره‌گرد استفاده شده که توانسته در مقایسه با الگوریتم ترتیبی استاندارد سرعت را به اندازه ۲۳٫۶ برابر افزایش دهد. در این الگوریتم از دو روش کلونی مورچه متعدد و مورچه‌های موازی استفاده شده است. در روش کلونی مورچه متعدد، چندید کلونی مورچه بین واحدهای پردازشی پخش می‌شوند و به هر کلونی یک بلاک اختصاص می‌یابد و برای هر مورچه یک نخ از همان بلاک در نظر گرفته می‌شود. در روش مورچه‌های موازی از یک کلونی مورچه استفاده می‌شود، مورچه‌ها بین عناصر پردازشی توزیع می‌شوند و برای هر مورچه یک نخ در نظر گرفته شده که عملیات محاسباتی مربوط به آن را انجام می‌دهد. در همه روش‌های ارائه‌شده برای موازی‌سازی و اجرای الگوریتم‌های مبتنی بر جمعیت در پردازنده گرافیکی، کارایی و سرعت اجرای الگوریتم‌ها به طور چشم‌گیری افزایش یافته که این مطلب اهمیت و توانایی پردازنده‌های گرافیکی را در اجرای الگوریتم‌های بهینه‌سازی نشان می‌دهد. در این تحقیق یک نسخه موازی از الگوریتم بهینه‌سازی غذایابی باکتری طراحی و پیاده‌سازی شده است که قابلیت اجرا بر روی پردازنده گرافیکی را دارد. این طراحی توسط بستر نرم‌افزاری کودا انجام شده که قابلیت اجرا در پردازنده‌های گرافیگی شرکت انویدیا را دارا می‌باشد. سپس الگوریتم ارائه‌شده با نام الگوریتم موازی غذایابی باکتری (CB-BFO) توسط چندین مسئله معروف بهینه‌سازی پیوسته با الگوریتم بهینه‌سازی غذایابی باکتری استاندارد مقایسه شد.

در ادامه به بررسی الگوریتم غذایابی باکتری و کارهای انجام‌شده قبلی می‌پردازیم. همچنین در بخش بعدی الگوریتم موازی غذایابی باکتری معرفی می‌شود. در بخش پایانی نتیجه مقایسه کارایی الگوریتم جدید با الگوریتم غذایابی باکتری استاندارد بررسی خواهد شد.

۲- پیشینه تحقیق

مسایل بهینه‌سازی دسته مهمی از مسایل هستند که در بیشتر شاخه‌های علوم کاربرد دارند. در این گونه مسایل هدف پیداکردن مقدار کمینه یا بیشینه (بهینه) از یک تابع یا فرایند است. برای حل این گونه مسایل الگوریتم‌های مختلفی ارائه شده است اما در چند دهه اخیر طبیعت الهام‌بخش ابداع تعداد نسباتاً زیادی از الگوریتم‌های توانمند بهینه‌سازی بوده است. از جمله قابلیت‌های این نوع الگوریتم‌ها می‌توان به توانایی جستجوی مؤثر فضاهای بسیار بزرگ در زمان کم، عدم نیاز به مشتق تابع هدف، توانایی گریز از نقاط بهینه محلی، هزینه محاسباتی بسیار کم و ریاضیات آسان اشاره کرد. یکی از این الگوریتم‌ها الگوریتم بهینه‌سازی غذایابی باکتری می‌باشد [۹]. این الگوریتم نیز یک الگوریتم مبتنی بر هوش تجمعی است که از روش غذایابی باکتری‌ها در طبیعت الهام گرفته است. با توجه به وجود ذرات مصنوعی یعنی همان باکتری‌ها در الگوریتم بهینه‌سازی غذایابی باکتری، امکان موازی‌سازی آن و طراحی برای اجرا در پردازنده گرافیکی وجود دارد.

۲-۱ الگوریتم بهینه‌سازی غذایابی باکتری

الگوریتم بهینه‌سازی غذایابی باکتری یکی از روش‌های بهینه‌سازی مبتنی بر جمعیت است که اولین بار در سال ۲۰۰۱ توسط پاسینو برای حل مسایل بهینه‌سازی ابداع و در سال ۲۰۰۲ به جامعه علمی معرفی شد [۹]. تاکنون از این الگوریتم برای حل مسایل مختلف مهندسی از جمله طراحی کنترل‌کننده‌های PID [۱۰]، کنترل تطبیقی [۹]، تخمین هارمونی [۱۱] و آموزش شبکه‌های عصبی [۱۲] با موفقیت استفاده شده است.

۲-۱-۱ نحوه غذایابی باکتری‌های تاژک‌دار در طبیعت

الگوریتم غذایابی باکتری با الهام‌گیری از زندگی برخی از باکتری‌های موجود در طبیعت از جمله ایکولای و سالامونلا که دارای ابزاری موسوم به تاژک برای حرکت‌دادن خود در محیط پیرامونشان هستند، ابداع شده است. این گونه از باکتری‌ها بسته به نوع چرخش تاژک خود به دو طریق می‌توانند در محیط پیرامونشان حرکت کنند. در اکثر باکتری‌های تاژک‌دار چرخاندن تاژک در خلاف جهت حرکت عقربه‌های ساعت منجر به حرکت رو به جلو و چرخاندن در جهت حرکت عقربه‌های ساعت باعث جنبش بی‌هدف باکتری و قرارگرفتن آن در یک مسیر تصادفی جدید می‌گردد. بدین ترتیب ترکیب جنبش تصادفی و حرکت رو به جلو باعث می‌شود که باکتری بتواند با جستجوی تصادفی در جهات مختلف به طور بهینه و مؤثری دنبال غذا بگردد. برای این منظور معمولاً اگر یک باکتری احساس کند که در حال نزدیک‌شدن به یک منبع غذایی غنی‌تر است با چرخاندن تاژک خود در خلاف جهت حرکت عقربه‌های ساعت رو به جلو حرکت می‌کند ولی اگر این باکتری بخواهد با تغییر جهت حرکت خود از منبع غذایی فعلی دور شده و به دنبال منابع غذایی بهتر بگردد تاژک خود را در جهت حرکت عقربه‌های ساعت می‌چرخاند. در عمل، مسیر حرکت باکتری‌ها ناشی از ترکیب پیچیده‌ای از شنای رو به جلو و جنبیدن تصادفی است که نتیجه آن قرارگرفتن هر باکتری در نقطه‌ای با بیشترین تراکم مواد غذایی می‌باشد. در علم زیست‌شناسی به این حرکت باکتری‌ها اصطلاحاً کموتکسیز گفته می‌شود. با توجه به توضیحات فوق می‌توان گفت که از نظر ریاضی عمل کموتکسیز در واقع نوعی جستجوی محلی برای یافتن موقعیت نقطه بهینه است. باکتری‌های تاژک‌دار مرتباً از عمل کموتکسیز برای یافتن منابع غذایی بهتر استفاده می‌کنند. در زندگی باکتری‌ها علاوه بر کموتکسیز دو عامل مهم دیگر نیز وجود دارد که عبارتند از هماوری یا تولید مثل و مرگ. همان طور که می‌دانیم هر باکتری برای انجام تولید مثل پس از گذشت زمان معینی از وسط به دو نیم تقسیم می‌شود. بدیهی است که در طبیعت باکتری‌هایی که دسترسی

مناسب‌تری به منابع غذایی داشته باشند از سرعت تولید مثل بالاتری نیز برخوردار خواهند بود. همچنین هر باکتری پس از گذشت زمان معینی می‌میرد و از گردونه حیات خارج می‌شود.

۲-۱-۲ شرح الگوریتم بهینه‌سازی غذایابی باکتری

در الگوریتم غذایابی باکتری نیز از روشی مشابه با روش غذایابی باکتری‌ها در طبیعت برای حل مسایل بهینه‌سازی استفاده می‌شود. در این الگوریتم ابتدا جمعیت اولیه‌ای از باکتری‌های مصنوعی که در ادامه تعداد آنها را با S نشان می‌دهیم به صورت تصادفی تولید شده و سپس در طول اجرای الگوریتم به ترتیب سه نوع عملیات بر روی آنها انجام می‌شود که عبارتند از کموتکسیز، تولید مثل و حذف- پراکندگی.

در هر مرحله کموتکسیز ابتدا هر یک از این باکتری‌ها به طور تصادفی در یک جهت خاص به حرکت درمی‌آید که به این حرکت در جهت تصادفی، اصطلاحاً لغزش و ادامه در جهت مسیر قبلی رانش گفته می‌شود. اگر در اثر این حرکت تصادفی باکتری به نقطه بهینه‌تری جابه‌جا شود این حرکت در مسیر قبلی ادامه پیدا می‌کند و در غیر این صورت حرکت باکتری متوقف می‌شود. پیش از انجام عمل تولید مثل که مرحله بعدی الگوریتم است، عمل کموتکسیز توسط تمامی باکتری‌ها به دفعات معین و از پیش تعیین شده که در ادامه با Nc نمایش می‌دهیم، انجام می‌شود.

پس از انجام عمل کموتکسیز مرحله تولید مثل آغاز می‌شود. در این مرحله باکتری‌ها بر اساس مجموع امتیازاتی که در طی Nc بار انجام عمل کموتکسیز کسب کرده‌اند مرتب‌سازی می‌شوند. پس از آن نیمی از باکتری‌ها که حایز بدترین امتیاز شده‌اند حذف می‌شوند و به جای هر یک از آنها یک کپی از هر یک از باکتری‌های باقی‌مانده جایگزین می‌گردد. بنابراین بر خلاف قوانین حاکم بر طبیعت که در آنها باکتری‌ها تا زمانی که شرایط مورد نیاز برای حیاتشان فراهم باشد به طور نامحدود به تکثیر و ازدیاد جمعیت ادامه می‌دهند، در الگوریتم غذایابی باکتری جمعیت باکتری‌ها با حذف نمونه‌های ضعیف‌تر و جایگزین‌کردن آنها با نمونه‌های قوی‌تر در طول اجرای الگوریتم ثابت نگه داشته می‌شود. پس از انجام عمل تولید مثل دوباره عمل کموتکسیز به همان ترتیبی که در قبل به آن اشاره شد انجام می‌شود و این فرایند تولید مثل نیز Nre بار تکرار می‌گردد. در واقع با توجه به توضیحات فوق در هر یک از Nre باری که عمل تولید مثل انجام می‌شود باید Nc بار عمل کموتکسیز انجام گیرد.

پس از Nre بار انجام عمل تولید مثل که هر یک مستلزم اجرای Nc بار انجام عمل کموتکسیز است، نوبت به مرحله پایانی یعنی مرحله حذف- پراکندگی می‌رسد. در این مرحله هر یک از باکتری‌ها با احتمال معین Ped دچار مرگ و سپس پراکندگی می‌گردند. برای این منظور اگر یک باکتری با مرگ خود از گردونه حیات حذف گردد به جای آن یک باکتری به صورت تصادفی در نقطه‌ای از دامنه مسئله قرار داده می‌شود. مرحله حذف- پراکندگی Ned بار انجام می‌گیرد. توجه به این نکته لازم است که الگوریتم غذایابی باکتری در واقع شامل سه حلقه تو در تو است به طوری که به ازای هر یک از Ned بار عمل حذف- پراکندگی، Nre بار عمل تولید مثل و به ازای هر یک از Nre بار عمل تولید مثل، Nc بار عمل کموتکسیز انجام می‌گیرد. به عبارت دیگر بیرونی‌ترین حلقه مربوط به عمل حذف- پراکندگی و داخلی‌ترین حلقه مربوط به عمل کموتکسیز است. وجود این سه حلقه تو در تو یکی از دلایل افزایش بار محاسباتی الگوریتم غذایابی باکتری در مقایسه با دیگر الگوریتم‌های بهینه‌سازی مبتنی بر جمعیت است.

در ادامه فرض می‌کنیم که $\theta^i(j,k,l)$ برداری n بعدی است که n تعداد متغیرها یا ابعاد مسئله است و این بردار موقعیت باکتری iام را در کموتکسیز jام، تولید مثل kام و حذف- پراکندگی lام نشان می‌دهد. در الگوریتم غذایابی باکتری در هر بار انجام عمل کموتکسیز موقعیت باکتری iام به صورت زیر تغییر داده می‌شود

$$\theta^i(j+1,k,l)=\theta^i(j,k,l)+c(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i).\Delta(i)}} \qquad (1)$$

که $\Delta(i)$ یک بردار n بعدی است که هر یک از درایه‌های آن یک عدد تصادفی با توزیع یکنواخت در بازه $[-1,1]$ می‌باشد و $i=1,2,\ldots,S$ است. ضریب $C(i)$ نیز اندازه گام کموتکسیز است که معمولاً عدد کوچکی مثل ۰/۱ یا ۰/۰۱ در نظر گرفته می‌شود. توجه به این نکته لازم است که در حالت کلی به هر یک از باکتری‌ها می‌توان اندازه گام متفاوتی را نسبت داد که استفاده از اندیس i نیز به همین منظور است. با این حال در اکثر مواقع تمام باکتری‌ها با گام‌هایی با اندازه یکسان و ثابت حرکت می‌کنند.

با توجه به توضیحات فوق، الگوریتم غذایابی باکتری را می‌توان به صورت زیر پیاده‌سازی کرد:

۱- مقداردهی اولیه: به هر یک از پارامترهای S، Nc، Ns، Nre، Ned، Ped و $C(i)$ که در آن:

S: تعداد باکتری‌ها

Nc: تعداد مراحل کموتکسیز

Ns: حداکثر تعداد رانش‌های متوالی برای هر باکتری در هر مرحله از کموتکسیز

Nre: تعداد مراحل تولید مثل

Ned: تعداد مراحل حذف- پراکندگی

Ped: احتمال حذف باکتری‌ها

$C(i)$: اندازه گام کموتکسیز باکتری iام

است، مقدار مناسبی نسبت دهید. به هر یک از باکتری‌ها یک مقدار تصادفی اولیه را از دامنه تعریف مسئله نسبت دهید.

۲- حلقه حذف- پراکندگی $l=l+1$

۳- حلقه تولید مثل $k=k+1$

۴- حلقه کموتکسیز $j=j+1$

۴-۱ به ازای $i=1,2,\ldots,S$ عمل کموتکسیز باکتری iام را به صورت زیر انجام دهید.

۴-۲ مقدار تابع هزینه یعنی $J(i,j,k,l)$ را محاسبه کنید.

۴-۳ قرار دهید $J_{Last}=J(i,j,k,l)$. دلیل ذخیره‌سازی $J(i,j,k,l)$ این است که بعداً این مقدار را با مقدار به دست آمده برای تابع هزینه توسط همین باکتری در کموتکسیز بعدی مقایسه خواهیم کرد.

۴-۴ لغزش: بردار تصادفی $\Delta(i)$ را که هر یک از درایه‌های آن یک عدد تصادفی با توزیع یکنواخت و در بازه $[-1,1]$ است تولید کنید.

۴-۵ حرکت: موقعیت باکتری را با استفاده از رابطه زیر به روز رسانی کنید

$$\theta^i(j+1,k,l)=\theta^i(j,k,l)+c(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i).\Delta(i)}} \qquad (2)$$

۴-۶ مقدار $J(i,j+1,k,l)$ را با استفاده از $\theta^i(j+1,k,l)$ محاسبه کنید.

۴-۷ شناکردن: مراحل زیر را انجام دهید:

۴-۷-۱ مقدار m شمارنده طول شنا را برابر صفر قرار دهید.

۴-۷-۲ تا زمانی که $m<Ns$ بود مراحل زیر را تکرار کنید:

۴-۷-۲-۱ قرار دهید $m = m+1$

۴-۷-۲-۲ اگر $J(i,j+1,k,l) < J_{Last}$ آن گاه J_{Last} را برابر $J(i,j+1,k,l)$ قرار دهید و مقدار جدید $\theta^i(j+1,k,l)$ را با استفاده از معادله موقعیت باکتری محاسبه کرده و با استفاده از مقدار به دست آمده برای $\theta^i(j+1,k,l)$، مقدار جدید $J(i,j+1,k,l)$ را محاسبه نمایید.

۴-۷-۲-۳ در غیر این صورت قرار دهید $m = Ns$ و حلقه ایجادشده توسط ۴-۷-۲ را پایان دهید.

۴-۸ به سراغ باکتری $i+1$ بروید. برای این منظور اگر $i < S$ است به مرحله ۴-۲ رفته و عملیات مربوط به باکتری بعدی را آغاز کنید.

۵- اگر $j < Nc$ آن گاه به مرحله ۴ بروید.

۶- تولید مثل: مراحل زیر را انجام دهید:

۶-۱ به ازای k و l داده‌شده و به ازای $i = 1,2,\ldots,S$، شاخص سلامت باکتری iام را با استفاده از معادله زیر محاسبه کنید

$$J^i_{Health} = \sum_{J=1}^{Nc+1} J(i,j,k,l) \qquad (۳)$$

۶-۲ باکتری‌ها را با استفاده از مقدار به دست آمده برای J^i_{Health}ها به ترتیب صعودی مرتب کنید.

۶-۳ تعداد Sr باکتری که $Sr = S/2$ را که حایز بیشترین مقدار برای J^i_{Health} شده‌اند از گردونه حیات حذف کنید و به جای هر یک از آنها یک کپی از Sr باکتری باقی‌مانده قرار دهید.

۷- اگر $k < Nre$ آن گاه به مرحله ۳ بروید.

۸- حذف- پراکندگی: به ازای $i = 1,2,\ldots,S$ هر یک از باکتری‌ها را با احتمال معین Ped حذف و سپس پراکنده کنید. برای این منظور در صورت حذف هر باکتری، یک باکتری جدید را در یک نقطه تصادفی از دامنه تعریف مسئله ایجاد نمایید. بدین ترتیب تعداد باکتری‌های کلونی ثابت باقی خواهد ماند.

۹- اگر $l < Ned$ به مرحله ۲ بروید.

۱۰- پایان الگوریتم

۲-۲ کارهای مرتبط انجام‌شده

بزرگ‌ترین مشکل الگوریتم غذایابی باکتری سرعت پایین همگرایی این الگوریتم در مقایسه با الگوریتم‌های معروف بهینه‌سازی دیگر مانند الگوریتم ژنتیک یا الگوریتم زنبور عسل است. همچنین الگوریتم غذایابی باکتری با توجه به سرعت پایین همگرایی در مواجهه با مسایل پیچیده بهینه‌سازی دچار مشکل شده و گاهاً توانایی پیداکردن جواب را ندارد. علت اصلی این مشکلات را می‌توان در بدنه الگوریتم مشاهده کرد. وجود سه حلقه تودرتو و همچنین حلقه باکتری‌ها که در داخل این سه حلقه اجرا می‌شود بار محاسباتی این الگوریتم را به شدت بالا می‌برد. این موضوع در هنگامی محسوس و مشخص است که از این الگوریتم برای حل مسایلی با ابعاد بالا و پیچیدگی زیاد استفاده شود.

کارهای زیادی برای حل این مشکل اساسی الگوریتم غذایابی باکتری انجام شده است. یکی از این راه حل‌ها ترکیب الگوریتم غذایابی باکتری با الگوریتم‌های قوی‌تر و استفاده از نقاط قوت این الگوریتم‌ها و ساخت یک الگوریتم هیبریدی است. برای مثال در [۱۳] با ترکیب الگوریتم غذایابی باکتری با الگوریتم تراکم ذرات یک الگوریتم هیبریدی طراحی شده که تا حدود زیادی مشکلات الگوریتم غذایابی باکتری را از لحاظ سرعت همگرایی پوشش می‌دهد. این الگوریتم در مقایسه با الگوریتم غذایابی باکتری و همچنین الگوریتم تراکم ذرات سرعت و کارایی بهتری را در حل چند مسئله بهینه‌سازی معروف از خود نشان داده است. همچنین در [۱۰] یک الگوریتم هیبریدی از الگوریتم غذایابی باکتری و الگوریتم ژنتیک ارائه شده است. این الگوریتم برای حل مسایل جستجو و همچنین برای تنظیم کنترل‌کننده‌های PID در رگلاتور ولتاژ اتوماتیک مورد استفاده قرار گرفته و کارایی بهتری را در مقایسه با الگوریتم‌های دیگر برای حل مسایل بهینه‌سازی از خود نشان داد. استفاده از ترکیب الگوریتم غذایابی باکتری با سیستم‌های فازی نیز مورد توجه محققان بوده است. در [۱۱] یک الگوریتم غذایابی باکتری فازی ارائه شده که برای انتخاب بهینه برای مرحله حرکت باکتری‌ها از استنتاج استفاده می‌کند. این الگوریتم برای تخمین هماهنگی سیگنال‌های تخریب‌شده توسط نویز مورد استفاده قرار گرفته است. در این الگوریتم اندازه حرکت باکتری متناسب با محیط عملیاتی مشخص می‌شود که این عمل باعث بالارفتن سرعت همگرایی الگوریتم شده است.

یکی دیگر از راه حل‌های بالابردن کارایی الگوریتم غذایابی باکتری که مورد توجه محققان قرار گرفته است، تغییر در ساختار الگوریتم برای ایجاد پارامترهای پویا است به طوری که الگوریتم قادر باشد در حین اجرا پارامترهای خود را تغییر داده و کارایی را بالا ببرد. در [۱۴] یک نسخه تغییریافته از الگوریتم غذایابی باکتری ارائه شده که کارایی بالاتری نسبت به الگوریتم استاندارد دارد. در این روش که روش انطباقی نام‌گذاری شده است، الگوریتم در هنگام اجرا و حل مسایل به صورت پویا اقدام به تغییر پارامترهای خود می‌کند. با استفاده از این روش سرعت همگرایی برای جواب‌دهی به چند تابع معیار در مقایسه با الگوریتم استاندارد و حتی الگوریتم‌هایی مانند الگوریتم ژنتیک و الگوریتم تراکم ذرات افزایش یافته است ولی این روش برای همه مسایل بهینه‌سازی مناسب نخواهد بود.

با مطرح‌شدن تکنیک‌های پردازش موازی و قابلیت‌های سخت‌افزاری پردازنده‌های چندهسته‌ای، ایده اجرای الگوریتم‌ها به صورت موازی مورد بررسی قرار گرفته است. الگوریتم غذایابی باکتری نیز توسط متخصصان برای اجرا در محیط‌های چندپردازنده‌ای تغییر یافته و توانسته است با اجرای به صورت موازی کارایی و سرعت قابل توجهی از خود نشان دهد. در [۱۵] یک الگوریتم موازی غذایابی باکتری ارائه شده که برای بهینه‌سازی در فشرده‌سازی تصاویر استفاده شده است. یکی از مراحل مهم فشرده‌سازی تصاویر، تخمین میزان حرکت تصاویر است که یک مسئله بهینه‌سازی می‌باشد. با استفاده از الگوریتم‌های بهینه‌سازی می‌توان این فرایند را به خوبی انجام داد ولی با توجه به سرعت همگرایی پایین در الگوریتم غذایابی باکتری استاندارد، از رهیافت موازی‌سازی برای بالابردن سرعت استفاده شده است. این الگوریتم در یک معماری چندپردازنده مورد استفاده قرار می‌گیرد. در این روش محاسبه بهینه‌ترین مقدار تابع تناسب به صورت هم‌زمان و در هنگامی که باکتری‌ها در مرحله حرکت قرار دارند انجام می‌شود. با استفاده از این الگوریتم به میزان ۹۶٫۶٪ سرعت محاسبات افزایش یافته است. همچنین در [۱۶] یک رهیافت موازی از الگوریتم غذایابی باکتری برای اجرا در معماری پردازنده‌های چندهسته‌ای ارائه شده است. مسئله‌ای که در این مقاله به آن اشاره شده یک مسئله بهینه‌سازی به نام ثبت تصاویر است که هدف آن مقایسه هم‌پوشانی دو یا چند تصویر از یک موضوع که در زمان‌های مختلف یا از نقطه دیدهای مختلف گرفته شده، می‌باشد. در این روش مرحله حرکت باکتری‌ها به طور هم‌زمان و موازی با محاسبه میزان تابع هدف در هر باکتری توسط پردازنده چندهسته‌ای انجام شده و ارتباط بین هسته‌ها در این روش توسط حافظه اشتراکی انجام می‌گردد. با استفاده از این الگوریتم می‌توان از توان واقعی پردازنده‌های چندهسته‌ای برای حل مسایل بهینه‌سازی پیچیده استفاده کرد.

۳- الگوریتم غذایابی باکتری موازی

الگوریتم غذایابی باکتری نیز مانند همه الگوریتم‌های مبتنی بر جمعیت از ذرات مصنوعی که همان باکتری‌ها هستند بهره می‌برد. در همه الگوریتم‌های مبتنی بر جمعیت این ذرات مصنوعی هستند که وظیفه جستجوی فضای مسئله را بر عهده دارند. در حالت معمولی جستجوی این ذرات به صورت ترتیبی و یکی‌یکی انجام می‌شود. دلیل آن هم در اختیار گرفتن پردازنده از طرف یکی از ذرات برای پردازش است. در صورت وجود امکان پردازش موازی، می‌توان پردازش بیش از یک ذره مصنوعی را در یک زمان و به صورت موازی انجام داد. در حقیقت این همان ماهیت پردازش موازی الگوریتم‌های مبتنی بر جمعیت است که در صورت موجودبودن محیط چندپردازنده‌ای می‌توان آنها را به صورت موازی و با سرعت بالاتر اجرا کرد. الگوریتم غذایابی باکتری نیز با توجه به وجود ذارت مصنوعی یعنی باکتری‌ها برای جستجو در آن می‌تواند در یک محیط چندپردازنده و به صورت موازی اجرا شود. در صورت طراحی یک الگوریتم کارامد و توجه به محدودیت‌های سخت‌افزاری می‌توان کارایی و سرعت این الگوریتم را با اجرای موازی تا چندین برابر بالا برد.

یکی از محیط‌های سخت‌افزاری مناسب برای اجرای موازی الگوریتم‌ها پردازنده گرافیکی است. پردازنده گرافیکی با استفاده از تعداد بالای هسته و سرعت محاسبات زیاد، سرعت اجرای الگوریتم ها را به شدت بالا می‌برد. الگوریتم موازی غذایابی باکتری ارائه‌شده با توجه به سخت‌افزار پردازنده گرافیکی طراحی شده و در این محیط قابل اجرا است.

۳-۱ بهره‌گیری از امکانات سخت‌افزار پردازنده گرافیکی و نرم‌افزار کودا

در طراحی الگوریتم ارائه‌شده موازی غذایابی باکتری از معماری کودا استفاده شده است. معماری کودا امکانات سخت‌افزاری و نرم‌افزاری مناسبی را جهت برنامه‌نویسی موازی به توسعه‌دهندگان ارائه می‌دهد که برای طراحی الگوریتم موازی غذایابی باکتری از آنها بهره برده‌ایم. برای این منظور به هر یک از باکتری‌ها یک نخ پردازشی اختصاص می‌دهیم.

۳-۱-۱ اجرای الگوریتم در یک بلاک

نخ‌های موجود در برنامه‌های کودا در قالب بلاک‌ها دسته‌بندی می‌شوند. نخ‌های موجود در یک بلاک می‌توانند با سرعت بیشتری با یکدیگر تبادل اطلاعات داشته باشند و الگوریتم موازی غذایابی باکتری نیز از این امکان بهره برده است. اجرای برنامه در داخل یک بلاک باعث شده که در مراحل مختلف اجرای الگوریتم به جای استفاده از حافظه سراسری بتوان از حافظه اشتراکی موجود در بلاک استفاده کرد. لازم به ذکر است که اجرای همه نخ‌ها در داخل یک بلاک محدودیت‌هایی نیز دارد که مهم‌ترین آنها محدودیت حداکثر نخ‌های موجود در داخل یک بلاک و همین طور محدودیت اندازه حافظه اشتراکی است.

۳-۱-۲ استفاده از قابلیت تولید اعداد تصادفی توسط هسته‌های پردازنده گرافیکی موجود در کتابخانه CURAND

یکی از مواردی که در اجرای الگوریتم موازی غذایابی باکتری نقش اساسی دارد، تولید اعداد تصادفی است. از اعداد تصادفی برای تعیین موقعیت باکتری‌های اولیه، احتمال حذف باکتری‌ها، تولید بردار تصادفی حرکت باکتری و جایگزینی باکتری‌های جدید استفاده می‌شود. در پردازنده گرافیکی با توجه به وجود تعداد هسته‌های متعدد، سرعت تولید اعداد تصادفی به مراتب بالاتر از پردازنده مرکزی می‌باشد. همچنین در صورت تولید اعداد تصادفی توسط پردازنده مرکزی نیاز است که این اعداد به حافظه اصلی پردازنده گرافیکی انتقال یابد که این موضوع باعث افت سرعت اجرا به دلیل تأخیر انتقال از حافظه اصلی به حافظه پردازنده گرافیکی خواهد شد. ولی با استفاده از تولید اعداد تصادفی در داخل پردازنده گرافیکی هم سرعت تولید اعداد بالاتر می‌رود و هم از تأخیر ناشی از انتقال اطلاعات جلوگیری می‌شود. در الگوریتم موازی غذایابی باکتری با استفاده از توابع کتابخانه تولید اعداد تصادفی موجود در معماری کودا، اعداد تصادفی مورد استفاده در داخل برنامه توسط پردازنده گرافیکی ساخته و مورد استفاده قرار می‌گیرد.

۳-۱-۳ استفاده از حافظه اشتراکی

حافظه اشتراکی، یک حافظه با سرعت بالا ولی حجم پایین است که در داخل هر بلاک و مختص به آن بلاک قرار دارد. با استفاده از این حافظه، نخ‌های موجود در هر بلاک می‌توانند با سرعت بیشتری به تبادل اطلاعات بپردازند. در صورت عدم استفاده از حافظه اشتراکی برای تبادل اطلاعات بین نخ‌ها نیاز است که از حافظه اصلی پردازنده گرافیکی استفاده شود که این حافظه علی‌رغم حجم بالا دارای سرعت بسیار پایین‌تر از حافظه اشتراکی است. لازم به ذکر است در صورت نیاز به تبادل اطلاعات بین نخ‌های موجود در بلاک‌های مختلف استفاده از حافظه سراسری یا حافظه اصلی پردازنده گرافیکی اجتناب‌ناپذیر می‌باشد. در الگوریتم موازی غذایابی باکتری در بسیاری موارد از حافظه اشتراکی استفاده می‌شود. مثلاً در زمان مرتب‌سازی باکتری‌ها نیاز است که شاخص سلامت باکتری‌ها با هم مقایسه شود که استفاده از حافظه اشتراکی سرعت این امر را بالا می‌برد یا در مرحله کموتکسیز همیشه نیاز است که مقدار تابع هدف در مرحله قبلی ذخیره و سپس با مقدار آن در مرحله جاری مقایسه شود. همچنین برای محاسبه مقدار تابع هدف و یا محاسبه نقطه حرکت بعدی باکتری به مقادیر قبلی موقعیت و جهت حرکت باکتری نیاز می‌باشد. ذخیره این مقادیر در داخل حافظه اشتراکی، سرعت دسترسی‌های آتی را برای آنها به طور محسوسی بالا می‌برد.

۳-۱-٤ استفاده از توابع ریاضی و تجزیه‌ناپذیر Atomic Function در کودا

در معماری کودا امکانات بسیار خوبی برای استفاده از توابع ریاضی و محاسباتی وجود دارد. همچنین وجود توابع تجزیه‌ناپذیر که اجرای آنها فقط در یک پالس ساعت انجام می‌شود، مزیت بسیار خوبی در این معماری است. در الگوریتم موازی غذایابی باکتری برای محاسبه مقدار توابع هدف از توابع ریاضی موجود در معماری کودا استفاده شده است. این امر نیاز به استفاده از توابع موجود در زبان C که برای اجرای آنها نیاز به پردازش توسط پردازنده مرکزی است را از بین برده است و باعث افزایش سرعت اجرای الگوریتم شده است. همچنین استفاده از یک تابع تجزیه‌ناپذیر که توانایی جمع و جابه‌جایی اعداد ممیز شناور در معماری کودا را دارد موجب افزایش سرعت برنامه شده است.

۳-۲ تغییرات انجام‌شده روی الگوریتم غذایابی باکتری

با توجه به ماهیت موازی الگوریتم غذایابی باکتری و وجود امکانات سخت‌افزاری و نرم‌افزاری موجود در معماری کودا، تغییراتی در ساختار الگوریتم غذایابی باکتری به وجود آوردیم که اجرای الگوریتم موازی غذایابی باکتری را به صورت موازی و در معماری کودا فراهم آورد. این تغییرات به شرح زیر است:

```
BITONIC SORT(Arr, d)
START
For i = 0 to d − 1
  For j = i downto 0
    If Arr[i+1] ≠ Arr[j] then
      Comp_exchange max(j);
    Else
      Comp_exchange min(j);
END
```

شکل ۱: شبه‌کد الگوریتم موازی مرتب‌سازی بایتونیک.

۳-۲-۱ مقداردهی اولیه و توزیع تصادفی باکتری‌ها در فضای مسئله به صورت موازی

مرحله مقداردهی اولیه در الگوریتم موازی غذایابی باکتری به صورت موازی و برای همه باکتری‌ها به طور هم‌زمان انجام می‌شود. این امر با کمک تولید اعداد تصادفی در داخل پردازنده گرافیکی و به صورت موازی و همچنین مقداردهی موقعیت همه باکتری‌ها و پخش آنها در فضای تعریف‌شده مسئله به صورت هم‌زمان انجام می‌پذیرد. برای این منظور از چند تابع موازی تعریف‌شده در داخل کرنل که عملیات تولید اعداد تصادفی، مقداردهی اولیه موقعیت باکتری‌ها، مقداردهی شاخص سلامت و تولید بردار تصادفی حرکت باکتری‌ها را به عهده دارند به صورت هم‌زمان برای همه باکتری‌ها استفاده گردید.

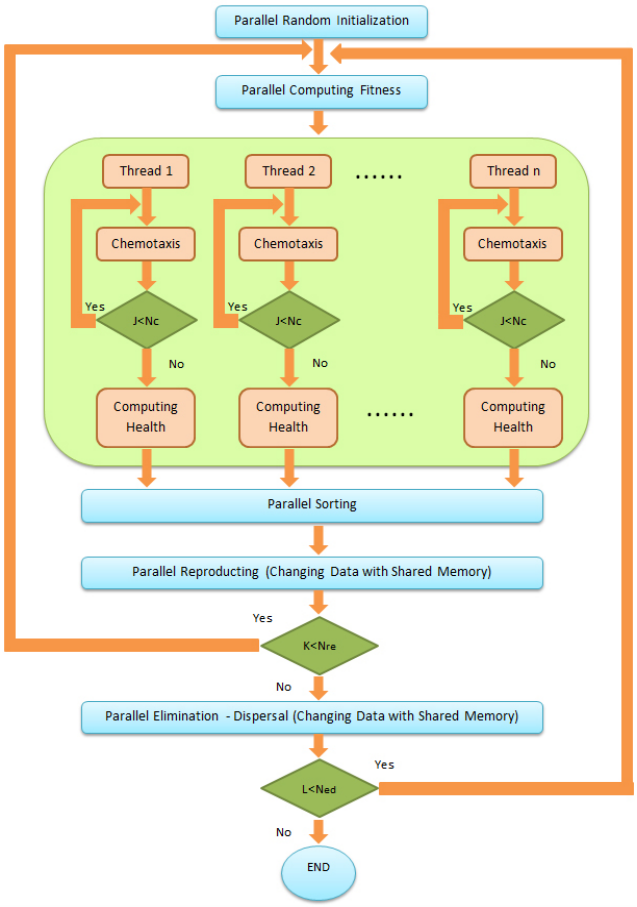
۳-۲-۲ محاسبه تابع هدف به صورت موازی

یکی از پردازش‌های مهم و پیچیده در داخل الگوریتم موازی غذایابی باکتری محاسبه تابع هدف می‌باشد. در الگوریتم غذایابی باکتری این عملیات به صورت ترتیبی و برای همه باکتری‌ها یکی پس از دیگری انجام می‌شود ولی در الگوریتم موازی غذایابی باکتری با استفاده از حافظه اشتراکی و بهره‌گیری از توابع تعریف‌شده ریاضی در معماری کودا این عملیات به صورت موازی و هم‌زمان برای همه باکتری‌ها انجام می‌شود. همچنین مقادیر به دست آمده برای تابع هدف در داخل حافظه اشتراکی ذخیره می‌شود تا دسترسی به آنها در مراحل بعدی با سرعت بیشتری انجام پذیرد.

۳-۲-۳ مرتب‌سازی باکتری‌ها به صورت موازی

یکی از مراحلی که در تمامی الگوریتم‌های مبتنی بر جمعیت وجود داشته و بار محاسباتی قابل توجهی را به خود اختصاص می‌دهد، مرحله مرتب‌سازی ذرات است. در الگوریتم غذایابی باکتری این مرتب‌سازی بر اساس شاخص سلامت باکتری و توسط روش‌های ترتیبی انجام می‌شود. در الگوریتم موازی غذایابی باکتری برای مرتب‌سازی از الگوریتم موازی مرتب‌سازی بایتونیک استفاده گردیده است. روش مرتب‌سازی بایتونیک از ۲ مرحله اصلی تشکیل شده است. ابتدا دنباله اعداد به یک توالی بایتونیک تبدیل شده و سپس این دنباله بایتونیک مرتب می‌شود. با استفاده از تعداد بالای هسته‌های پردازنده گرافیکی می‌توان این مرتب‌سازی را به صورت موازی انجام داد. شبه‌کد الگوریتم مرتب‌سازی بایتونیک در شکل ۱ آمده است. در این شبه‌کد Arr آرایه ورودی اعداد برای مرتب‌سازی است و تعداد اعداد برابر n در نظر گرفته شده که $n=2^d$ که d در حقیقت اندازه توان عدد ۲ است.

در الگوریتم موازی غذایابی باکتری برای پیاده‌سازی الگوریتم مرتب‌سازی بایتونیک به صورت موازی از ساختار ابرمکعب استفاده شده است.



شکل ۲: روندنمای الگوریتم موازی غذایابی باکتری.

۳-۲-۴ تولید مثل باکتری‌ها به صورت موازی

در الگوریتم غذایابی باکتری پس از مرتب‌سازی باکتری‌ها توسط شاخص سلامت، نیمه بد باکتری‌ها حذف شده و نیمه خوب آنها تولید مثل می‌کنند. این عمل در الگوریتم غذایابی باکتری به صورت ترتیبی و برای هر یک از باکتری‌ها پی در پی انجام می‌شود ولی در الگوریتم موازی غذایابی باکتری با استفاده از نخ‌های پردازشی این عملیات، موازی و هم‌زمان برای همه باکتری‌های تولید مثل کننده و حذف‌شونده انجام می‌شود. با توجه به وجود اطلاعات مربوط به باکتری در داخل حافظه اشتراکی تغییرات لازم بر روی اطلاعات باکتری سریع‌تر انجام می‌پذیرد.

۳-۲-۵ حذف و پراکندگی باکتری‌ها به صورت موازی

برای حذف باکتری‌ها در الگوریتم غذایابی باکتری به هر یک از باکتری‌ها یک عدد تصادفی برای حذف نسبت داده می‌شود و با مقایسه این عدد با پارامتر Ped نسبت به بقا یا حذف باکتری تصمیم‌گیری می‌شود. در صورت حذف باکتری، یک باکتری جدید به صورت تصادفی در فضای مسئله جایگزین آن می‌گردد. در الگوریتم موازی غذایابی باکتری عملیات مقداردهی عدد تصادفی برای باکتری‌ها توسط توابع کرنل به صورت موازی انجام می‌گیرد. سپس به صورت هم‌زمان مقدار این عدد تصادفی برای همه باکتری‌ها مقایسه شده و در صورت حذف باکتری، یک باکتری جایگزین در فضای مسئله قرار می‌گیرد. ذخیره عدد تصادفی هر باکتری و مقایسه آن با پارامتر Ped در داخل حافظه اشتراکی و به صورت موازی انجام می‌پذیرد که این امر سرعت اجرا را بالا می‌برد.

با توجه به توضیحات ذکرشده می‌توان روندنمای الگوریتم موازی غذایابی باکتری را به صورت شکل ۲ در نظر گرفت. در این شکل کلیه

جدول ۱: سخت‌افزار مورد استفاده برای شبیه‌سازی.

Device	CPU	GPU۱	GPU۲	GPU۳
Processor	Intel i۷-۴۷۹۰	GeForce GTX ۷۶۰	GeForce GTX ۶۵۰	GeForce GT ۶۳۰
Number of Core	۸	۱۱۵۲	۳۸۴	۹۶
Clock	۳٫۶۰ GHz	۱٫۰۷ GHz	۱٫۰۵ GHz	۰٫۷ GHz
Memory Type	DDR۳ ۱۶۰۰ MHz	DDR۵ ۳۰۰۴ MHz	DDR۵ ۲۶۰۰ MHz	DDR۳ ۱۸۰۰ MHz
Memory Size	۱۶ GB	۲ GB	۱ GB	۱ GB
OS	Win ۸.۱ (۶۴ – bit)	–	–	–
Compute Capability Version	–	۳٫۰	۳٫۰	۲٫۱
CUDA Version	–	۶٫۵	۶٫۵	۶٫۵

جدول ۲: توابع محک برای ارزیابی الگوریتم موازی غذایابی باکتری.

Function		Equation	Domain
F۱	Rastrigin	$f(x)=۱۰n+\sum_{i=۱}^{n}[x_i^۲-۱۰\cos(۲\pi x_i)]$	$-۵٫۱۲<x_i<۵٫۱۲$
F۲	Rosenbrock	$f(x)=\sum_{i=۱}^{n-۱}[۱۰۰(x_{i+۱}-x_i^۲)^۲+(x_i-۱)^۲]$	$-۱۰<x_i<۱۰$
F۳	Michalewicz	$f(x)=-\sum_{i=۱}^{n}\sin x_i[\sin\frac{ix_i^۲}{\pi}]^{۲۰}$	$۰<x_i<\pi$
F۴	Schwefel	$f(x)=۴۱۸٫۹۸۲۹n-\sum_{i=۱}^{n}x_i\sin\sqrt{\lvert x_i\rvert}$	$-۵۰۰<x_i<۵۰۰$

نخ‌ها که هر کدام معرف یک باکتری هستند، ابتدا به صورت موازی مقداردهی اولیه شده و مقدار تابع تناسب آنها محاسبه می‌شود. سپس کلیه نخ‌ها در داخل یک بلاک عملیات کموتکسیز را به صورت موازی انجام می‌دهند. پس از آن مراحل بعدی الگوریتم به صورت موازی برای همه نخ‌ها انجام می‌شود.

٤- بررسی کارایی الگوریتم موازی غذایابی باکتری

پس از ارائه هر روش یا الگوریتم جدید، کارایی آن باید مورد بررسی و تحقیق قرار گیرد. ساده‌ترین و شاید معمول‌ترین روش تعیین کارایی الگوریتم، مقایسه آن با الگوریتم‌های مشابه قبلی است. در مورد الگوریتم‌های بهینه‌سازی مبتنی بر جمعیت، معمولاً کارایی الگوریتم جدید را برای حل مسایل بهینه‌سازی استاندارد و شناخته‌شده در مقایسه با دیگر الگوریتم‌ها می‌سنجند. توابع محک از این دسته مسایل بهینه‌سازی پیوسته هستند که معمولاً برای مقایسه الگوریتم‌ها استفاده می‌شوند. در این فصل الگوریتم موازی غذایابی باکتری برای حل چند مسئله بهینه‌سازی استاندارد یعنی تعدادی از توابع محک با الگوریتم سریال استاندارد غذایابی باکتری و همچنین الگوریتم موازی اجراشده روی پردازنده Multicore مقایسه شده و نتایج اجرای الگوریتم‌ها برای مقایسه و تعیین کارایی الگوریتم جدید ثبت می‌گردد.

٤-۱ ابزارهای نرم‌افزاری مورد استفاده شبیه‌سازی

یکی از مزایای مهم برنامه‌نویسی در کودا پشتیبانی کامل زبان C از آن می‌باشد. با توجه به این موضوع بهترین محیط برای توسعه نرم‌افزارهای مبتنی بر کودا، نرم‌افزار Visual C++.NET می‌باشد که کلیه فایل‌ها و کتابخانه‌های مورد نیاز برای برنامه‌نویسی کودا را فراهم می‌کند. در این تحقیق نیز ما از نسخه ۲۰۱۰ این نرم‌افزار برای برنامه‌نویسی الگوریتم موازی غذایابی باکتری استفاده کرده‌ایم. همچنین از نسخه ۶٫۵ بستر نرم‌افزاری کودا برای طراحی الگوریتم موازی غذایابی باکتری استفاده شده و برای مقایسه بین الگوریتم موازی غذایابی باکتری (CB-BFO) و الگوریتم سریال استاندارد غذایابی باکتری (BFO)، یک نرم‌افزار نیز برای الگوریتم سریال غذایابی باکتری طراحی شد. برای این منظور نیز از زبان C و محیط برنامه‌نویسی Visual C++.NET استفاده شد تا شرایط نرم‌افزاری مشابهی را با الگوریتم موازی غذایابی باکتری داشته باشد. همچنین یک برنامه موازی برای اجرای الگوریتم غذایابی باکتری (MC-BFO) در محیط Multicore طراحی گردید که کارایی الگوریتم موازی غذایابی باکتری با این الگوریتم نیز مقایسه شود که برای این منظور نیز از زبان C و محیط برنامه نویسی Visual C++.NET استفاده شد. این برنامه از چندین هسته قابل دسترس در پردازنده مشخص‌شده در جدول ۱ استفاده می‌کند.

٤-۲ سخت‌افزار مورد استفاده

برای اجرای هر سه الگوریتم موازی غذایابی باکتری، الگوریتم سریال استاندارد غذایابی باکتری و الگوریتم اجراشده در محیط چندهسته‌ای از سخت‌افزار یکسانی مطابق جدول ۱ استفاده شد. همچنین برای اطمینان از کارایی الگوریتم موازی غذایابی باکتری اجرای آن بر روی ۳ سخت‌افزار GPU مورد بررسی قرار گرفت که مشخصات این سخت‌افزارها نیز در جدول ۱ آمده است.

٤-۳ معرفی توابع محک

برای مقایسه بین الگوریتم‌های مبتنی بر جمعیت معمولاً از توابع شناخته‌شده محک استفاده می‌شود، بدین صورت که با استفاده از هر کدام از الگوریتم‌ها نسبت به حل یک یا چند تابع محک که از مسایل شناخته‌شده بهینه‌سازی هستند اقدام می‌شود. سپس نتیجه اجرای الگوریتم‌ها را برای حل هر کدام از مسایل با یکدیگر مقایسه می‌کنند. مسایل و توابع استاندارد مختلفی برای آزمون این الگوریتم‌ها معرفی شده است [۱۷]. در این تحقیق ما از ۴ تابع معروف محک که از پیچیده‌ترین مسایل می‌باشند استفاده کرده‌ایم که مشخصات آنها در جدول ۲ آمده است. در حل این توابع محک توسط الگوریتم‌های مبتنی بر جمعیت هیچ

جدول ۳: پارامترهای تعیین‌شده برای اجرای الگوریتم‌های استاندارد غذایابی باکتری و موازی غذایابی باکتری و برنامه چندهسته‌ای.

Function	*Nc*	*Ned*	*Nre*	*Ped*	*Ci*	*Ns* (in BFO)	*S* (Fixed *S*)	Dimension (Fixed Dimension)
F۱	۱۰۰	۴	۸	۰/۲۵	۰/۰۵	۶	۶۴	۲
F۲	۱۵۰	۶	۱۰	۰/۲۵	۰/۰۱	۱۰	۶۴	۲
F۳	۱۵۰	۶	۱۰	۰/۲۵	۰/۰۱	۱۲	۶۴	۲
F۴	۱۰۰	۶	۸	۰/۲۵	۰/۰۱	۱۰	۶۴	۲

گاه به جواب دقیق دست پیدا نمی‌کنیم و جواب‌ها به صورت تقریبی می‌باشند و بنابراین برای به دست آوردن جواب بهینه نیاز به تعریف حدی برای حداکثر خطا در جواب‌های این توابع داریم. در این مقاله حداکثر خطای در نظر گرفته شده بین جواب دقیق و جواب‌های به دست آمده برای توابع محک ۰/۰۰۱ می‌باشد. با توجه به ثابت‌بودن تعداد اجرا در الگوریتم غذایابی باکتری، موقعیت‌های تصادفی تعریف‌شده برای توابع محک تأثیر زیادی در سرعت اجرای الگوریتم ندارد و فقط بر روی مقدار خطای جواب نهایی به دست آمده تأثیرگذار می‌باشد و بنابراین مبنای پذیرش جواب‌های به دست آمده کمتربودن خطای جواب از مقدار مشخص‌شده است.

٤-٤ روش ارزیابی

برای ارزیابی الگوریتم جدید به دو روش متفاوت عمل گردید. ابتدا پارامترهای مناسب برای اجرای هر کدام از مسایل یا توابع محک انتخاب شد. نحوه انتخاب این پارامترها کارهای قبلی محققان [۱۸]، [۱۳] و [۹] و روش آزمون و خطا می‌باشد. این پارامترها در جدول ۳ آمده است.

در روش اول ابتدا تعداد ابعاد متغیرهای مسئله ثابت و برابر ۲ در تمامی توابع در نظر گرفته شد و بقیه پارامترها برای هر یک از توابع مطابق جدول ۳ قرار داده شد. سپس تعداد باکتری‌ها برابر عددی از توان ۲ و از ۴ تا ۱۰۲۴ متغیر قرار داده شد و در هر حالت، هر کدام از الگوریتم‌های سریال غذایابی باکتری (BFO) و الگوریتم موازی غذایابی باکتری (CB-BFO) و برنامه چندهسته‌ای غذایابی باکتری (MC-BFO) ۳۰ بار اجرا شد و زمان میانگین ۳۰ بار اجرا برای هر کدام از الگوریتم‌ها ثبت گردید.

در روش دوم تعداد باکتری‌ها در تمامی توابع برابر ۶۴ و بقیه پارامترها مطابق جدول ۳ در نظر گرفته شد. سپس ابعاد مسئله یا تعداد متغیرها از ۲ تا ۱۰ قرار داده شد و در هر حالت، هر کدام از الگوریتم‌های سریال غذایابی باکتری (BFO) و الگوریتم موازی غذایابی باکتری (CB-BFO) و برنامه چندهسته‌ای غذایابی باکتری (MC-BFO) ۳۰ بار اجرا شد و زمان میانگین ۳۰ بار اجرا برای هر کدام از الگوریتم‌ها ثبت گردید. در هر اجرا در صورتی که مقدار خطای به دست آمده بیش از مقدار مشخص شده بود آن جواب از مقدارهای به دست آمده حذف شده و برنامه بار دیگر اجرا گردید. همچنین برای اطمینان از کارایی الگوریتم موازی غذایابی باکتری اجرای این الگوریتم بر روی سه سخت‌افزار مختلف GPU مورد بررسی قرار گرفت که برای این منظور نیز ابعاد متغیرها برابر ۲ و تعداد باکتری‌ها از ۴ تا ۱۰۲۴ متغیر قرار داده شد و در این حالت نیز میانگین زمان ۳۰ بار اجرای الگوریتم بر روی هر یک از سخت‌افزارها ثبت گردید. در جدول ۴ زمان‌های ۳۰ بار اجرا برای حالتی که ابعاد متغیرها برابر ۲ و تعداد باکتری‌ها برابر ۲۵۶ می‌باشد آمده است.

٤-٥ نتایج مقایسه و نمودارها

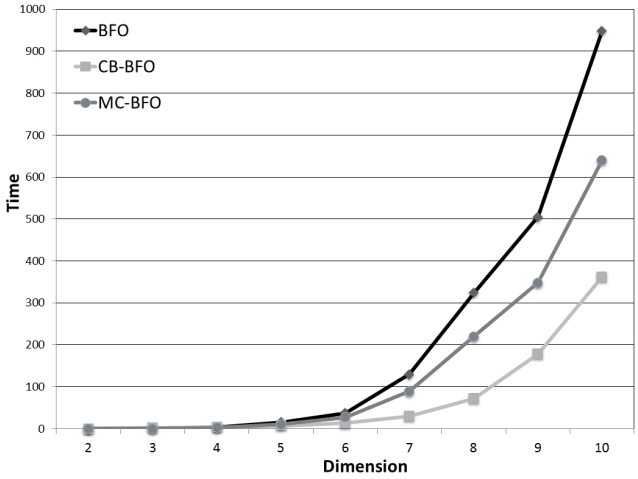
در روش اول تعداد ابعاد مسئله ثابت و تعداد باکتری‌ها از ۴ تا ۱۰۲۴ به صورت توانی از عدد ۲ قرار داده شد. میانگین زمان‌های به دست آمده برای اجرای هر الگوریتم در جدول ۵ آمده است. در این روش میانگین افزایش سرعت الگوریتم موازی غذایابی باکتری نسبت به الگوریتم استاندارد غذایابی باکتری در حدود ۴/۴۳ برابر ثبت شد. همچنین حداکثر افزایش سرعت در تابع F۳ و در تعداد باکتری ۱۰۲۴ اتفاق افتاد که افزایش سرعت در حدود ۹/۱۴ برابر بود. در روش دوم تعداد باکتری‌ها ثابت و تعداد ابعاد مسئله از ۲ تا ۱۰ متغیر قرار داده شد. میانگین زمان‌های به دست آمده در جدول ۶ آمده است. در این روش میانگین افزایش سرعت الگوریتم موازی غذایابی باکتری نسبت به الگوریتم استاندارد غذایابی باکتری در حدود ۳/۸۹ برابر ثبت شد. همچنین حداکثر افزایش سرعت در تابع F۳ و با تعداد متغیر ۷ اتفاق افتاد که افزایش سرعت در حدود ۱۱/۱۹ برابر بود. لازم به ذکر است در این حالت الگوریتم استاندارد غذایابی باکتری در سه مورد قادر به محاسبه جواب نشد که یکی در تابع F۱ و در ابعاد ۱۰ مسئله و دو مورد دیگر در تابع F۳ و در ابعاد ۹ و ۱۰ مسئله بود. همچنین با اجرای الگوریتم غذایابی باکتری در محیط چندهسته‌ای مشخص شد که به دلیل محدودیت و ثابت‌بودن تعداد هسته‌های CPU سرعت اجرای الگوریتم با یک نرخ ثابت افزایش می‌یابد ضمن این که در این محیط سخت‌افزاری نیز الگوریتم قادر به حل مسئله در برخی از شرایط نمی‌باشد که این مشکل در الگوریتم استاندارد سریال غذایابی باکتری نیز وجود داشت. با انجام هر سه روش، مجموع زمان‌های اجرا برای هر یک از توابع و افزایش سرعت الگوریتم موازی جدید نسبت به الگوریتم سریال استاندارد غذایابی باکتری مشخص گردید که این نتایج در جدول ۷ آمده است. میانگین افزایش سرعت الگوریتم موازی غذایابی باکتری به نسبت الگوریتم استاندارد غذایابی باکتری در حدود ۴/۱۶ برابر ثبت گردید.

نمودار متوسط زمان اجرای هر سه الگوریتم در حالتی که تعداد باکتری‌ها ثابت و تعداد ابعاد مسئله متغیر است در شکل ۳ آمده است. همچنین شکل ۴ نمودار متوسط زمان اجرای هر سه الگوریتم را در حالتی که ابعاد مسئله ثابت بوده و تعداد باکتری‌ها متغیر هستند نشان می‌دهد. همچنین بعد از اجرای الگوریتم موازی غذایابی باکتری در سه GPU مختلف نتایج این اجراها در شکل ۵ ثبت گردیده که مشخص می‌کند تغییر قابلیت‌های سخت‌افزاری GPU بر روی سرعت اجرای الگوریتم تأثیر مستقیم دارد که دلیل آن تغییر تعداد هسته‌های پردازشی GPUها و همچنین تفاوت در قابلیت پردازشی و اندازه حافظه اشتراکی در هر یک از GPUها می‌باشد.

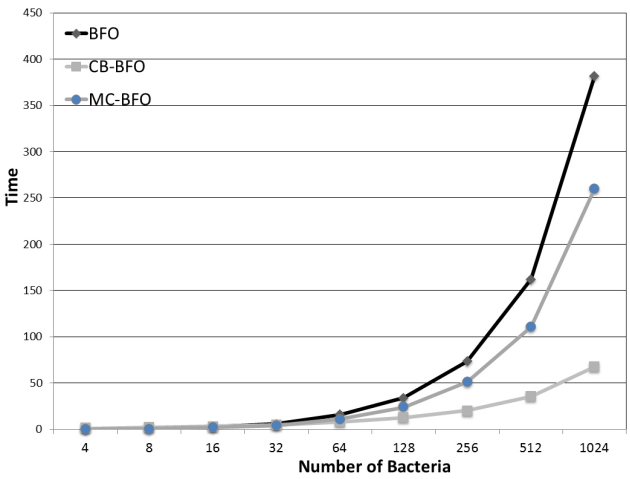
همچنین میزان استفاده از GPU و CPU، در هنگام اجرای الگوریتم سریال استاندارد غذایابی باکتری در شکل ۶ مشخص شده و این مقادیر هنگام اجرای الگوریتم موازی غذایابی باکتری نیز در شکل ۷ آمده است.

۵- نتیجه‌گیری

در این مقاله ابتدا به معرفی الگوریتم بهینه‌سازی غذایابی باکتری



شکل ۳: نمودار میانگین زمان اجرای سه الگوریتم در حالت ثابت‌بودن تعداد باکتری‌ها.

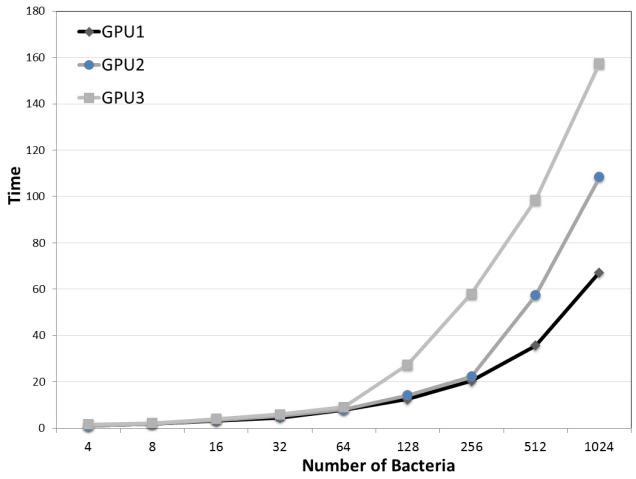


شکل ۴: نمودار میانگین زمان اجرای سه الگوریتم در حالت ثابت‌بودن تعداد ابعاد مسئله.

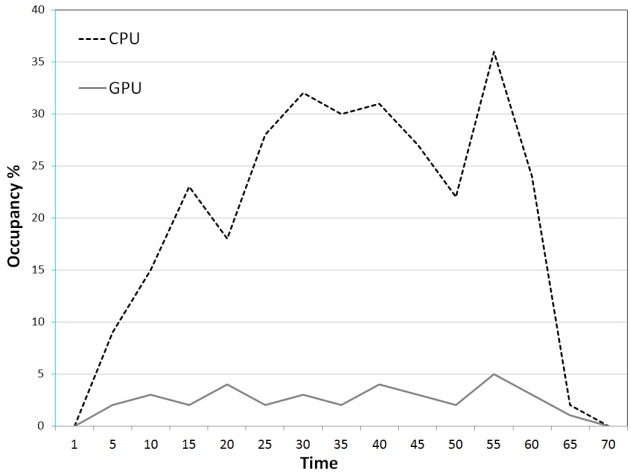
جدول ۴: زمان‌های به دست آمده برای ۳۰ اجرای الگوریتم موازی غذایابی باکتری برای حالت ابعاد متغیرها ۲ و تعداد باکتری‌ها ۲۵۶.

شماره اجرا	F۱			F۲			F۳			F۴		
	BFO	MC-BFO	CB-BFO	BFO	MC-BFO	CB-BFO	BFO	MC-BFO	CB-BFO	BFO	MC-BFO	CB-BFO
۱	۴۷٫۸۵	۳۰٫۵۶	۱۴٫۲۳	۹۲٫۶۱	۵۲٫۳۶	۳۱٫۴۲	۱۱۵٫۶۵	۷۰٫۸۹	۱۶٫۶۹	۵۵٫۶۱	۳۶٫۴۷	۲۴٫۹۶
۲	۵۶٫۴۷	۳۲٫۴۱	۱۵٫۳۶	۸۵٫۵۴	۵۲٫۵۷	۲۷٫۹۹	۱۱۲٫۴۵	۶۶٫۳۴	۱۳٫۸۸	۴۷٫۵۲	۳۴٫۶۹	۲۵٫۶۲
۳	۴۳٫۵۶	۳۷٫۴۳	۱۲٫۲۵	۸۱٫۳۲	۶۵٫۴۲	۲۳٫۸۹	۹۷٫۸۹	۶۹٫۳۴	۱۶٫۱۴	۴۷٫۶۹	۴۰٫۴۷	۲۲٫۸۱
۴	۶۲٫۱۴	۳۷٫۰۹	۱۳٫۴۷	۹۰٫۰۶	۶۲٫۰۱	۲۶٫۰۲	۹۵٫۶۳	۷۴٫۱۵	۱۴٫۹۵	۵۵٫۴۱	۳۸٫۲۲	۲۵٫۵۲
۵	۴۷٫۹۲	۳۵٫۲۲	۱۱٫۲۳	۸۸٫۱۴	۶۷٫۳۷	۳۲٫۳۸	۱۰۱٫۴۵	۷۵٫۶۷	۱۳٫۶۳	۵۸٫۴۶	۴۳٫۲۵	۲۷٫۳۸
۶	۵۰٫۲۳	۳۷٫۰۴	۱۵٫۱۹	۸۷٫۳۲	۶۶٫۴۶	۲۴٫۷۲	۱۱۴٫۵۶	۶۶٫۴۹	۱۵٫۴۷	۵۳٫۴۴	۴۱٫۱۱	۲۵٫۲۳
۷	۵۸٫۷۷	۴۰٫۹۶	۱۴٫۱۲	۹۳٫۴۶	۶۸٫۳۸	۲۹٫۶۱	–	۷۱٫۲۳	۱۲٫۳۶	۴۷٫۷۱	۳۳٫۲۴	۲۴٫۸۱
۸	۴۱٫۱۲	۲۷٫۴۵	–	۸۰٫۸۶	۵۵٫۴۶	۲۳٫۷۴	۱۱۶٫۹۴	۷۴٫۱۴	۱۴٫۰۸	۴۸٫۰۹	۳۸٫۷۵	۲۴٫۹۷
۹	۴۰٫۱۸	۲۹٫۶۳	۱۳٫۳۵	۸۷٫۰۶	۵۹٫۴۳	۲۳٫۴۴	۹۸٫۱۶	۷۴٫۰۸	۱۵٫۲۵	۵۹٫۸۵	۳۸٫۳۶	۲۱٫۹۳
۱۰	۵۳٫۴۵	۳۰٫۰۶	۱۴٫۳۶	۹۲٫۵۹	۶۷٫۷۷	۳۳٫۹۵	۱۰۷٫۱۵	۶۴٫۱۱	۱۶٫۹۷	۴۸٫۶۵	۳۴٫۹۳	۲۷٫۸۱
۱۱	۵۲٫۳۶	۴۰٫۱۸	۱۵٫۴۷	۹۳٫۰۷	۶۳٫۲۲	۳۲٫۷۴	۱۱۸٫۰۶	۶۹٫۶۹	۱۵٫۸۷	۴۹٫۸۸	۳۵٫۶۴	۲۶٫۲۲
۱۲	–	۳۹٫۴۳	۱۱٫۰۹	۸۶٫۶۲	۵۸٫۰۲	۲۷٫۶۱	۱۰۲٫۴۵	۶۶٫۸۷	۱۳٫۹۱	۴۸٫۵۲	۳۶٫۰۹	۲۵٫۸۴
۱۳	۴۷٫۵۸	۳۶٫۳۲	۱۲٫۶۹	۱۰۰٫۳۶	۶۴٫۹۶	۲۴٫۰۶	۹۸٫۷۳	۷۰٫۰۲	۱۴٫۵۷	۵۲٫۴۸	۳۷٫۸۳	۲۲٫۹۵
۱۴	۴۹٫۳۷	۳۹٫۵۶	۱۴٫۴۴	۸۹٫۵۵	۶۵٫۳۲	–	۱۱۹٫۸۷	۷۰٫۵۸	۱۳٫۹۸	۵۷٫۷۷	۴۰٫۳۶	۲۸٫۰۸
۱۵	۴۶٫۳۷	۴۰٫۷۲	۱۲٫۲۷	۸۳٫۴۷	۵۱٫۳۶	۳۰٫۳۲	۹۳٫۰۹	۶۵٫۶۷	۱۲٫۵۵	۵۲٫۵۴	۳۸٫۴۵	۲۷٫۶۲
۱۶	۵۲٫۵۵	۳۴٫۶۷	۱۲٫۹۸	۸۰٫۳۹	۵۳٫۷۷	۲۳٫۱۴	۱۰۲٫۵۵	۷۰٫۹۲	۱۵٫۸۱	۵۷٫۶۵	۳۶٫۵۹	۲۷٫۴۸
۱۷	۵۸٫۷۳	۳۶٫۱۲	۱۵٫۱۶	۸۶٫۶۵	۵۹٫۴۱	۲۳٫۳۲	۱۰۷٫۴۶	۶۳٫۶۹	۱۵٫۹۳	۵۷٫۴۷	۴۳٫۱۳	۲۹٫۸۳
۱۸	۵۰٫۳۸	۳۹٫۲۸	۱۵٫۱۶	۹۰٫۶۲	۵۲٫۰۷	۲۹٫۴۶	۹۳٫۶۸	۷۳٫۶۶	۱۵٫۲۱	۴۷٫۵۵	۳۳٫۶۸	۲۷٫۵۱
۱۹	۵۱٫۹۳	۳۷٫۸۳	۱۱٫۷۱	۸۴٫۱۷	۶۶٫۷۴	۲۸٫۳۱	۱۰۱٫۷۹	۶۷٫۰۹	۱۵٫۰۵	۵۲٫۵۳	۴۳٫۷۱	۲۱٫۳۵
۲۰	۴۵٫۹۴	۳۶٫۷۴	۱۴٫۶۸	۸۶٫۵۹	۵۹٫۱۶	–	۸۹٫۴۳	۶۹٫۶۴	۱۵٫۶۸	۵۶٫۰۹	۳۵٫۹۸	۲۷٫۹۵
۲۱	۴۱٫۹۳	۳۸٫۴۴	۱۲٫۰۵	۹۱٫۴۳	۶۴٫۰۱	۲۷٫۵۲	۱۰۸٫۰۷	۶۵٫۷۱	۱۵٫۳۲	۵۳٫۲۵	۳۸٫۶۵	۲۶٫۱۹
۲۲	۴۷٫۰۵	۳۶٫۳۳	۱۳٫۲۸	۸۰٫۴۴	۶۰٫۹۲	۲۷٫۶۱	۹۲٫۶۱	۶۸٫۶۷	۱۳٫۸۵	۵۰٫۱۲	۴۳٫۶۳	۲۲٫۵۱
۲۳	۵۲٫۰۸	۳۲٫۲۷	۱۳٫۴۷	۸۸٫۹۸	۵۵٫۱۶	۲۹٫۱۳	۹۷٫۶۶	۷۳٫۷۲	–	۵۵٫۹۳	۴۰٫۱۷	۲۱٫۴۲
۲۴	۴۸٫۷۴	۳۴٫۸۷	۱۵٫۲۴	۸۶٫۸۷	۶۱٫۲۵	۳۲٫۷۴	۱۰۶٫۵۸	۷۴٫۹۲	۱۵٫۱۲	۵۶٫۹۲	۳۸٫۸۵	۲۳٫۰۷
۲۵	–	۳۹٫۴۷	۱۲٫۷۸	۹۰٫۷۶	۵۷٫۴۸	۲۴٫۵۲	۱۰۳٫۹۶	۷۰٫۵۸	۱۴٫۷۴	۵۷٫۳۱	۳۲٫۱۵	۲۵٫۲۴
۲۶	۴۹٫۷۱	۴۰٫۴۷	۱۴٫۹۱	۸۷٫۵۹	۵۱٫۹۶	۲۸٫۱۴	–	۷۴٫۹۲	۱۲٫۶۹	۴۶٫۲۶	۳۴٫۹۵	۲۸٫۷۲
۲۷	۵۷٫۶۳	۳۰٫۲۵	۱۲٫۷۷	۸۰٫۰۳	۶۲٫۵۶	۲۸٫۴۶	۹۰٫۲۲	۷۴٫۳۳	۱۲٫۵۷	۵۹٫۸۶	۳۲٫۵۶	۲۲٫۰۸
۲۸	۵۲٫۷۲	۳۶٫۸۵	۱۳٫۰۸	۸۱٫۶۱	۵۱٫۷۵	۳۳٫۶۱	۱۰۷٫۱۷	۷۰٫۵۱	۱۴٫۴۶	۵۲٫۳۵	۴۲٫۳۵	۲۹٫۴۵
۲۹	۴۲٫۰۵	۳۹٫۸۸	۱۲٫۵۶	۹۷٫۱۱	۵۴٫۸۴	۳۰٫۰۵	۹۶٫۷۲	۷۲٫۲۶	۱۱٫۸۹	۵۰٫۳۱	۴۲٫۰۳	۲۵٫۵۹
۳۰	۵۶٫۲۲	۳۴٫۹۵	۱۳٫۰۱	۸۱٫۲۵	۵۹٫۳۲	۲۲٫۵۹	۱۰۲٫۷۴	۶۴٫۲۸	۱۳٫۶۱	۵۹٫۷۴	۴۰٫۴۶	۲۹٫۲۹
میانگین	۵۰٫۱۸	۳۶٫۰۸	۱۳٫۵۳	۸۷٫۵۵	۵۹٫۶۸	۲۷٫۸۷	۱۰۳٫۳۱	۷۰٫۱۴	۱۴٫۵۶	۵۳٫۲۲	۳۸٫۲۵	۲۵٫۶۱

پرداختیم. سپس نقاط ضعف این الگوریتم را بیان نموده و کارهای قبلی انجام‌شده در این مورد را بررسی کردیم. در ادامه با استفاده از امکان موازی‌سازی الگوریتم غذایابی باکتری و همچنین امکانات نرم‌افزاری و سخت‌افزاری معماری کودا یک الگوریتم غذایابی باکتری موازی ارائه



شکل ۵: مقایسه میانگین زمان اجرا برای GPUهای مختلف.



شکل ۶: میزان Occupancy مربوط به CPU و GPU در زمان اجرای الگوریتم سریال استاندارد غذایابی باکتری.

جدول ۵: متوسط زمان اجرای الگوریتم سریال غذایابی باکتری استاندارد و الگوریتم موازی غذایابی باکتری و برنامه چندهسته‌ای در حالت ثابت‌بودن ابعاد مسئله.

Bacteria	F۱			F۲			F۳			F۴		
	BFO	MC-BFO	CB-BFO	BFO	MC-BFO	CB-BFO	BFO	MC-BFO	CB-BFO	BFO	MC-BFO	CB-BFO
۴	۰٫۰۹	۰٫۰۶	۰٫۶۳	۰٫۱۵	۴	۰٫۰۹	۰٫۰۶	۰٫۶۳	۰٫۱۵	۴	۰٫۰۹	۰٫۰۶
۸	۰٫۳۲	۰٫۲۵	۱٫۴۴	۱٫۵۲	۸	۰٫۳۲	۰٫۲۵	۱٫۴۴	۱٫۵۲	۸	۰٫۳۲	۰٫۲۵
۱۶	۱٫۲۷	۰٫۸۸	۲٫۵۳	۳٫۴۳	۱۶	۱٫۲۷	۰٫۸۸	۲٫۵۳	۳٫۴۳	۱۶	۱٫۲۷	۰٫۸۸
۳۲	۴٫۵۲	۳٫۱۲	۳٫۷۱	۶٫۵۲	۳۲	۴٫۵۲	۳٫۱۲	۳٫۷۱	۶٫۵۲	۳۲	۴٫۵۲	۳٫۱۲
۶۴	۱۴٫۳۲	۹٫۸۷	۶٫۳۲	۱۵٫۰۷	۶۴	۱۴٫۳۲	۹٫۸۷	۶٫۳۲	۱۵٫۰۷	۶۴	۱۴٫۳۲	۹٫۸۷
۱۲۸	۲۳٫۶۸	۱۷٫۰۲	۹٫۵۳	۳۶٫۸۰	۱۲۸	۲۳٫۶۸	۱۷٫۰۲	۹٫۵۳	۳۶٫۸۰	۱۲۸	۲۳٫۶۸	۱۷٫۰۲
۲۵۶	۵۰٫۱۸	۳۶٫۰۸	۱۳٫۵۳	۸۷٫۵۵	۲۵۶	۵۰٫۱۸	۳۶٫۰۸	۱۳٫۵۳	۸۷٫۵۵	۲۵۶	۵۰٫۱۸	۳۶٫۰۸
۵۱۲	۱۱۷٫۳۲	۷۹٫۸۲	۲۱٫۸۹	۱۸۹٫۴۷	۵۱۲	۱۱۷٫۳۲	۷۹٫۸۲	۲۱٫۸۹	۱۸۹٫۴۷	۵۱۲	۱۱۷٫۳۲	۷۹٫۸۲
۱۰۲۴	۲۶۹٫۴۷	۱۸۳٫۴۵	۴۵٫۰۴	۴۳۰٫۴۳	۱۰۲۴	۲۶۹٫۴۷	۱۸۳٫۴۵	۴۵٫۰۴	۴۳۰٫۴۳	۱۰۲۴	۲۶۹٫۴۷	۱۸۳٫۴۵

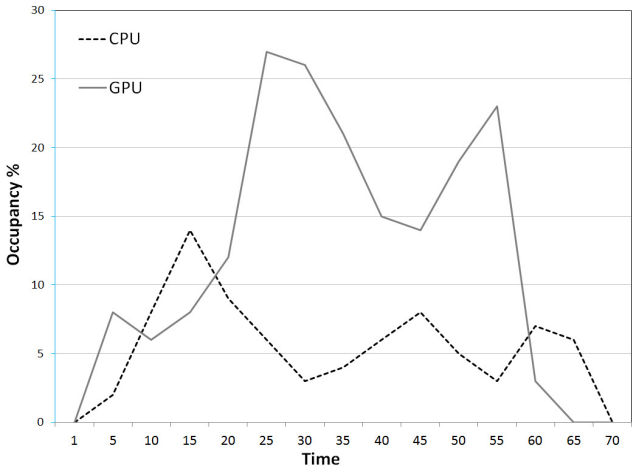
جدول ۶: متوسط زمان اجرای الگوریتم سریال غذایابی باکتری استاندارد و الگوریتم موازی غذایابی باکتری و برنامه چندهسته‌ای در حالت ثابت‌بودن تعداد باکتری‌ها.

Dimension	F۱			F۲			F۳			F۴		
	BFO	MC-BFO	CB-BFO	BFO	MC-BFO	CB-BFO	BFO	MC-BFO	CB-BFO	BFO	MC-BFO	CB-BFO
۲	۰٫۰۲	۰٫۰۱	۰٫۷۸	۰٫۰۵	۰٫۰۳	۰٫۶۷	۰٫۰۵	۰٫۰۳	۱٫۴۲	۰٫۰۷	۰٫۰۵	۰٫۹۳
۳	۰٫۷۵	۰٫۵۴	۲٫۷۱	۱٫۲۷	۰٫۸۶	۱٫۸۹	۱٫۲۷	۰٫۸۷	۲٫۸۹	۱٫۲۱	۰٫۸۲	۲٫۰۶
۴	۴٫۸۵	۳٫۴۲	۴٫۳۶	۳٫۱۸	۲٫۲۶	۲٫۶۵	۴٫۱۸	۲٫۸۱	۴٫۵۳	۳٫۴۵	۲٫۴۵	۳٫۷۴
۵	۹٫۶۳	۶٫۷۳	۸٫۹۲	۱۲٫۶۵	۸٫۵۶	۵٫۴۷	۲۶٫۷۴	۱۸٫۳۶	۶٫۷۶	۹٫۶۵	۶٫۷۸	۶٫۱۹
۶	۲۵٫۱۷	۱۷٫۳۴	۱۹٫۹۸	۲۷٫۸۳	۱۹٫۴۷	۹٫۱۶	۷۴٫۶۵	۵۱٫۳۲	۱۲٫۳۹	۲۱٫۵۳	۱۶٫۲۳	۱۴٫۶۲
۷	۹۷٫۲۲	۶۹٫۴۷	۳۶٫۷۷	۶۳٫۷۲	۴۳٫۷۲	۲۵٫۶۷	۲۸۷٫۳۶	۱۹۳٫۶۵	۲۵٫۶۷	۶۹٫۳۶	۴۸٫۲۵	۳۲٫۴۵
۸	۲۸۷٫۲۳	۲۰۱٫۳۲	۶۹٫۰۶	۱۸۹٫۱۴	۱۲۸٫۴۵	۶۸٫۲۴	۶۳۹٫۵۲	۴۲۹٫۵۷	۶۸٫۲۴	۱۷۶٫۵۶	۱۱۹٫۴۵	۷۹٫۲۲
۹	۶۴۷٫۰۹	۴۵۲٫۸۲	۱۸۹٫۳۶	۴۷۳٫۴۶	۳۱۹٫۵۲	۱۴۹٫۱۲	–	–	۱۷۸٫۴۲	۳۹۶٫۰۷	۲۷۴٫۹۸	۱۹۷٫۵۷
۱۰	–	–	۳۴۷٫۱۶	۹۳۶٫۱۲	۶۲۹٫۳۴	۳۱۴٫۹۳	–	–	۳۶۹٫۱۳	۹۵۸٫۶۳	۶۴۸٫۲۷	۴۱۲٫۲۲

جدول ۷: مجموع زمان‌های اجرای دو الگوریتم سریال استاندارد و موازی و میزان افزایش سرعت الگوریتم جدید برای هر یک از توابع.

Function	BFO	CB-BFO	Speedup
F۱	۱۵۵۳٫۱۳	۴۳۶٫۵۶	۳٫۵۶
F۲	۲۴۷۸٫۳۶	۷۶۰٫۴۳	۳٫۲۶
F۳	۱۹۵۵٫۸۷	۲۵۳٫۹۲	۷٫۷۰
F۴	۲۱۶۶٫۴۷	۹۴۶٫۱۳	۲٫۲۹

نمودیم که قابلیت اجرا بر روی سخت‌افزار پردازنده کارت گرافیکی را دارد. سپس با استفاده از چند تابع محک سرعت و کارایی الگوریتم جدید موازی غذایابی باکتری را با الگوریتم سریال غذایابی باکتری استاندارد و اجرای آن در محیط چندهسته‌ای مقایسه نمودیم. همان طور که در قسمت قبل ذکر شد الگوریتم موازی غذایابی باکتری افزایش سرعتی در حدود ۴٫۱۶ برابر الگوریتم استاندارد غذایابی باکتری ثبت کرد. این میزان افزایش سرعت در مقایسه با الگوریتم‌های موازی ارائه‌شده دیگر مبتنی بر جمعیت مانند الگوریتم زنبور با افزایش سرعت ۱۳ برابری [۶] و الگوریتم تراکم ذرات با افزایش سرعت ۱۱ برابری [۷] میزان قابل قبولی می‌باشد ضمن این که وجود سه حلقه تو در تو در بدنه این الگوریتم کارایی آن را برای حل مسایلی با پیچیدگی بالا کاهش داده است. بنابراین مشاهده شد که



شکل ۷: میزان Occupancy مربوط به CPU و GPU در زمان اجرای الگوریتم موازی غذایابی باکتری.

الگوریتم غذایابی باکتری در مواجهه با بعضی از مسایل پیچیده و در اینجا مشخصاً تابع F۳ در ابعاد بالا توانایی دست‌یافتن به جواب درست را ندارد. این در حالی است که الگوریتم موازی غذایابی باکتری توانسته است در تمامی موارد به جواب دست پیدا کند. لازم به ذکر است در ابعاد پایین مسایل و در تعداد کم باکتری‌ها سرعت اجرای الگوریتم غذایابی باکتری بالاتر از سرعت اجرای موازی غذایابی باکتری می‌باشد. دلیل این مورد تأخیر انتقال اطلاعات از حافظه اصلی به حافظه کارت گرافیکی و همچنین انتقال از حافظه اصلی کارت گرافیکی به حافظه اشتراکی موجود در بلاک است. در تعداد بالای باکتری‌ها و ابعاد بالاتر مسایل این زمان تأخیر جبران می‌شود.

با مشاهده شیب منحنی‌های زمان اجرای سه الگوریتم، مشخص می‌شود که با بالارفتن تعداد بعد مسایل شیب منحنی الگوریتم موازی غذایابی باکتری نیز مانند الگوریتم غذایابی باکتری به شدت بالا می‌رود و انتظار می‌رود در ابعاد بالاتر مسایل، الگوریتم موازی غذایابی باکتری نیز قادر به حل مسایل نباشد. این مسئله به دلیل وجود سه حلقه تو در تو در الگوریتم و بالارفتن بار محاسباتی با بالارفتن تعداد ابعاد مسئله اتفاق می‌افتد. ولی با بالارفتن تعداد باکتری‌ها شیب منحنی الگوریتم موازی غذایابی باکتری بسیار ملایم‌تر از شیب منحنی الگوریتم غذایابی باکتری می‌باشد. این مورد نشان می‌دهد که می‌توان تعداد باکتری‌های الگوریتم را از ۱۰۲۴ بالاتر برد. با بالابردن تعداد باکتری‌ها در عوض می‌توان پارامترهای دیگر الگوریتم را مانند *Nc* و *Ns* کاهش داد. کاهش این پارامترها باعث کاهش بار محاسباتی الگوریتم می‌شود و بدین ترتیب می‌توان تعداد ابعاد مسایل را باز هم افزایش داد. این بدان معنی است که با افزایش تعداد باکتری‌ها و تغییر در پارامترهای اصلی الگوریتم موازی غذایابی باکتری می‌توان سرعت اجرا را بالاتر برد و یا توانایی الگوریتم را برای اجرای مسایلی با ابعاد بالاتر افزایش داد. همچنین اجرای الگوریتم در محیط چندهسته‌ای و مقایسه آن با الگوریتم سریال استاندارد نشان داد به دلیل ثابت‌بودن تعداد هسته‌های CPU افزایش سرعت اجرای الگوریتم نرخ ثابتی را دنبال می‌کند ضمن این که مشکل کارایی الگوریتم حل نشده باقی ماند و در این حالت نیز الگوریتم قادر به حل مسئله در ابعاد بزرگ نبود.

اجرای الگوریتم موازی بر روی سه GPU مختلف نشان داد که سرعت اجرای الگوریتم با تعداد هسته‌های موجود در GPU نسبت مستقیم دارد. در GPU۲ با توجه به وجود ۳۸۴ هسته پردازشی مشاهده شد که سرعت اجرا در ابعاد زیر ۵۱۲ تفاوت چندانی با GPU۱ ندارد ولی در ابعاد ۵۱۲ به بعد کاهش سرعت محسوسی مشاهده می‌شود که دلیل آن کم‌بودن تعداد هسته‌ها و نیاز به اجرای الگوریتم در دو بلاک مجزا و در ابعاد ۱۰۲۴ در سه بلاک مجزا می‌باشد. همچنین در GPU۳ این کاهش سرعت به میزان خیلی کم در ابعاد پایین قابل مشاهده است که دلیل آن پایین‌بودن قابلیت پردازشی و کمتربودن میزان حافظه اشتراکی این GPU می‌باشد ولی در ابعاد ۱۲۸ به بالا به دلیل وجود ۹۶ هسته پردازشی سرعت اجرا به طور مشخص کاهش می‌یابد که دلیل آن استفاده از دو و بیشتر بلاک مجزا برای اجرای برنامه است. همچنین نمودار Occupancy برای CPU و GPU در دو حالت اجرای سریال و موازی نشان داد که در حالت سریال فقط CPU درگیر محاسبات می‌باشد و در این حالت پردازنده گرافیکی فقط نقش ایجاد تصویر را با پردازش بسیار پایین انجام می‌دهد که با توجه به سطوح زیر نمودار در شکل ۶ مقدار متوسط استفاده از CPU در این حالت ۲۱٫۱۵٪ و برای GPU ۲٫۵۶٪ می‌باشد. ولی در حالت موازی بیشتر بار محاسباتی به عهده GPU است و فقط در زمانی که اطلاعات بین GPU و GPUانتقال می یابد و یا در زمان تخصیص حافظه توسط CPU و همچنین در انتهای اجرای برنامه که نتایج از GPU به حافظه اصلی برگردانده می‌شوند، CPU درگیر محاسبات می‌شود. در زمان‌هایی که بار محاسباتی CPU افزایش یافته است، درگیری GPU در محاسبات کمتر شده و برعکس زمانی که GPU درگیر پردازش می‌شود میزان استفاده از CPU کاهش می‌یابد. در الگوریتم موازی با توجه به سطوح زیر نمودار در شکل ۷ متوسط استفاده از CPU ۵٫۷۷٪ و استفاده از GPU ۱۲٫۹۴٪ بوده است.

یکی از کارهایی که برای انجام در آینده مد نظر است موازی‌سازی و اجرای الگوریتم‌های ضعیف دیگر در پردازنده گرافیکی و با استفاده از معماری کودا می‌باشد. همچنین اجرای الگوریتم‌های هیبریدی در محیط پردازنده گرافیکی نیز از کارهای مورد نظر برای آینده می‌باشد. یکی دیگر از مواردی که می‌توان به عنوان تحقیقات آینده به آن اشاره کرد اجرای الگوریتم‌های بهینه‌سازی مبتنی بر جمعیت بر روی دو یا چند پردازنده گرافیکی می‌باشد که نیاز به سخت‌افزار خاص دارد.

مراجع

[1] L. M. Schmitt, "Theory of genetic algorithms," *Theoretical Computer Science Science Direct*, vol. 259, no. 1-2, pp. 1-61, 2001.

[2] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. on Neural Networks*, vol. 4, pp. 1942-1948, 27 Nov.-1 Dec. 1995.

[3] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies," in *Proc. European Conf. on Artificial Life Paris France*, vol. 142, pp. 134-142, 1991.

[4] T. Sato and M. Hagiwara, "Bee system: finding solution by a concentrated search," in *Proc. IEEE Int. Conf. on Systems Man and Cybernetics Computational Cybernetics and Simulation*, vol. 4, pp. 3954-3959, 12-15 Oct. 1997.

[5] P. Pospichal, J. Jaros, and J. Schwarz, "Parallel genetic algorithm on the CUDA architecture," *EvoApplications*, vol. 6024, no. 1, pp. 442-451, 2010.

[6] G. H. Luo, S. K. Huang, Y. S. Chang, and S. M. Yuan, "A parallel bee algorithm implementation on GPU," *J. of System Architecture*, vol. 60, no. 3, pp. 271-279, Mar. 2013.

[7] Y. Zhou and Y. Tan, "GPU-based parallel particle swarm optimization," in *Proc. IEEE Congress on Evolutionary Computation CEC'09*, pp. 1493-1500, 18-21 May 2009.

[8] A. Delevacq, P. Delisle, M. Gravel, and M. Krajecki, "Parallel ant colony optimization on graphics processing units," *J. of Parallel and Distributed Computing)*, vol. 73, no. 1, pp. 52-61, Jan. 2013.

[9] K. M. Passino, "Biomimicry of bacterial foraging for distributed optimization and control," *IEEE Control System Magazine*, vol. 22, no. 3, pp. 52-67, Jun. 2002.

[10] D. H. Kim, A. Abraham, and J. H. Cho, "A hybrid genetic algorithm and bacterial foraging approach for global optimization," *Information Science*, vol. 177, no. 18, pp. 3918-3937, 15 Sept. 2007.
[11] S. Mishra, "A hybrid least square-fuzzy bactrial foraging strategy for harmonic estimation," *IEEE Trans. on Evolutionary Computation*, vol. 9, no. 1, pp. 61-73, Feb. 2005.
[12] C. Cabrita, J. Botzheim, A. E. Ruano, and L. T. Koczy, "Genetic programing and bacterial algorithm for neural networks and fuzzy systems design," in *Proc. Conf. on Intelligent Control Systems and Signal Processing,* 6 pp., Faro, Portugal, 2003.
[13] A. Biswas, S. Dasgupta, S. Das, and A. Abraham, "Synergy of PSO and bacterial foraging optimization-a comparative study on numerical benchmarks," *Innovations in Hybrid Intelligent Systems*, vol. 44, no. 1, pp. 255-263, 2007.
[14] H. Chen, Y. Zhu, and K. Hu, "Adaptive bacterial foraging optimization," *Abstract and Applied Analysis*, vol. 2011, Article ID 108269, 27 pp., 2011.
[15] K. M. Bakwad, S. S. Pattnik, B. S. Shi, S. Devi, and M. R. Lohakare, "Parallel bacterial foraging optimization for video compression," *International J. of Recent Trends in Engineering*, vol. 1, no. 1, pp. 118-122, May 2009.
[16] S. I. Bejinariu, "Image registration using bacterial foraging optimization algorithm on multi-core processors," in *Proc. 4th Int. Symp. on Electrical and Electronics Engineering, ISEEE'13*, 6 pp., 11-13 Oct. 2013.
[17] M. Molga and C. Smutnicki, "Test functions for optimization needs," *J. of Computer Sciences and Applications Science and Education Publishing*, vol. 2, no. 2, pp. 23-30, 2005.
[18] H. Chen, Y. Zhu, and K. Hu, "Cooperative bacterial foraging optimization," *Discrete Dynamics in Nature and Society*, vol. 2009, Article ID 815247, 17 pp., 2009.

علی رفیعی در سال ۱۳۸۱ مدرک کارشناسی مهندسی کامپیوتر گرایش نرم افزار خود را از دانشگاه علم و فرهنگ تهران دریافت نمود. از سال ۱۳۸۵ تا سال ۱۳۹۲ به عنوان مدیر برنامه ریزی و انفورماتیک شرکت کیارابر مشغول به کار بود و پس از آن به دوره کارشناسی ارشد مهندسی کامپیوتر گرایش نرم‌افزار در دانشگاه آزاد اسلامی واحد اراک وارد گردید و در سال ۱۳۹۴ موفق به اخذ مدرک کارشناسی ارشد مهندسی کامپیوتر از دانشگاه آزاد اسلامی واحد اراک شد. درسال ۱۳۹۵ به دوره دکترای مهندسی کامپیوتر گرایش نرم افزار در دانشگاه آزاد اسلامی واحد اراک وارد گردید و هم‌اکنون به عنوان دانشجوی دکترا در این دانشگاه مشغول به تحصیل می‌باشد. زمینه‌های علمی مورد علاقه نام‌برده عبارتند از: الگوریتم‌های مبتنی بر جمعیت، الگوریتم‌های موازی و پردازش موازی.

سیدمرتضی موسوی در سال ۱۳۸۰ مدرک کارشناسی ارشد خود را در رشته معماری کامپیوتر از دانشگاه علوم و تحقیقات تهران اخذ نموده و در حال حاضر دانشجوی دکتری کامپیوتر این دانشگاه است. همچنین ایشان عضو هیأت علمی گروه کامپیوتر دانشگاه آزاد اراک می باشد. زمینه‌های تحقیقاتی ایشان عبارتند از: معماری کامپیوتر، پردازش موازی، شبکه‌های کامپیوتری، شبکه‌های حسگر بی‌سیم، امنیت شبکه‌های کامپیوتری، رمزنگاری داده‌ها.