

# موازی سازی الگوریتمهای ژنتیکی با استفاده از اسکلت‌های الگوریتمی

حسین دلداری\* و تکتم غفاریان\*\*

گروه مهندسی کامپیوتر، دانشگاه فردوسی مشهد

(دریافت مقاله: ۸۰/۱۲/۲۵ - دریافت نسخه نهایی: ۸۲/۴/۳۰)

**چکیده** - اسکلت‌های الگوریتمی<sup>۱</sup> به عنوان شیوه‌ای برای برنامه نویسی موازی در سالهای اخیر مورد توجه محققان قرار گرفته است. با استفاده از این شیوه، برنامه نویس می‌تواند با بهره‌گیری از یک سری الگوهای پیش ساخته، الگوریتم خود را پیاده سازی کند. در این مقاله مجموعه‌ای از الگوهای پیش ساخته با الهام از این شیوه برای پیاده سازی الگوریتمهای ژنتیکی موازی طراحی شده است. برای هر یک از الگوهای پیشنهادی، مدل کارایی<sup>۲</sup> محاسبه شده است. برنامه نویس با استفاده از این مدل، قادر به مقایسه الگوها و انتخاب بهترین الگو برای کاربرد مورد نظرش است. ضمناً برنامه نویس می‌تواند در هر الگو توپولوژی مجازی پردازشها را تعیین کند. این عمل ضمن افزایش کارایی الگو، کار جدیدی در تحقیقات اسکلت‌های الگوریتمی است. از دیگر دستاوردهای این تحقیق قابلیت ترکیب الگوهاست که تنها در معدودی از تحقیقات قبلی دیده شده است.

واژگان کلیدی: اسکلت الگوریتمی، الگوریتم ژنتیکی موازی، مدل کارایی، توپولوژی مجازی، مهاجرت

## Parallel Genetic Algorithm Using Algorithmic Skeleton

H. Deldari, T. Ghafarian

Computer Department, Ferdowsi University of Mashhad

**Abstract:** *Algorithmic skeleton has received attention as an efficient method of parallel programming in recent years. Using the method, the programmer can implement parallel programs easily. In this study, a set of efficient algorithmic skeletons is introduced for use in implementing parallel genetic algorithm (PGA). A performance model is derived for each skeleton that makes the comparison of skeletons possible in order to select the best one for the application. The performance of the selected skeleton can be increased by specifying the virtual topology required by the application. This is a novel approach with no precedent. Nesting of skeletons used here is another novelty of the study which has been employed only in few previous studies.*

**Keywords:** *Algorithmic Skeleton, Parallel Genetic Algorithm, Performance Model, Virtual Topology, Migration*

\*\* - کارشناسی ارشد

\* - استادیار

محاسبه مدل کارایی برای پیشگویی زمان اجرای هر یک از اسکلت‌های پیشنهادی آورده شده است. در بخش هفتم نتایج حاصل از پیاده سازی الگوهای پیشنهادی بر روی شبکه‌ای از رایانه‌های شخصی آمده است و در نهایت مقاله با نتیجه‌گیری و پیشنهاداتی برای تحقیقات آتی خاتمه می‌یابد.

## ۲- اسکلت الگوریتمی

اسکلت الگوریتمی، یک مدل کلی برای برنامه‌نویسی موازی است که توسط مری کل<sup>۱۲</sup> در سال ۱۹۸۹ معرفی شده است [۳]. هدف از آن ایجاد زبانی است که ضمن برنامه‌نویسی آسان و مستقل از معماری ماشین، برنامه نوشته شده نیز کارایی بالایی داشته باشد. اسکلت، یک الگوی الگوریتمی متداول در برنامه‌های موازی است [۳]. اسکلتها معمولاً در داخل یک زبان سری میزبان<sup>۱۳</sup> قرار می‌گیرند و تنها منبع موازی سازی در آن زبان محسوب می‌شوند. به عنوان مثال اسکلت نگاشت<sup>۱۴</sup>، تابعی را بر روی تمام عضوهای یک لیست به طور موازی اجرا می‌کند. اسکلت مزرعه<sup>۱۵</sup> ساختار ارباب - بنده<sup>۱۶</sup> را به طور موازی پیاده‌سازی می‌کند و اسکلت تقسیم و حل<sup>۱۷</sup> یک مسئله را با تقسیمات متوالی بازگشتی، به طور موازی حل می‌کند [۴].

مهمترین ویژگی یک اسکلت، کلی بودن آن است که عبارت از قابلیت استفاده از آن اسکلت در کاربردهای متفاوت است. از این رو در اغلب اسکلتها، مجموعه‌ای از توابع وجود دارد که باید توسط استفاده کننده اسکلت تعریف شود. پس از تعریف این توابع توسط کاربر، با انجام یک پیش پردازش این توابع در مکان مشخصی از اسکلت درج و سپس اجرا می‌شود. از جمله تحقیقات انجام شده در زمینه اسکلت‌های الگوریتمی می‌توان به کارهای دارلینگتون<sup>۱۸</sup> [۴]، پلاگاتی<sup>۱۹</sup> [۵] و رابهی<sup>۲۰</sup> [۶] اشاره کرد.

معمولاً هر اسکلت دارای یک مدل کارایی است که به صورت یک فرمول ریاضی ارائه می‌شود و کاربر با درج پارامترهای مربوط به مسئله خود در این مدل، قادر به پیشگویی زمان اجرای برنامه‌اش خواهد بود.

محاسبات علمی قرن حاضر نیاز به استفاده از رایانه‌هایی با کارایی بالا دارد. با توجه به محدودیت در افزایش سرعت رایانه‌های سری، چنین رایانه‌هایی می‌توانند رایانه‌های موازی باشند. رایانه‌های موازی امروزه دارای چندین هزار پردازشگراند. این رایانه‌ها دارای تکنولوژی بالایی بوده و از دسترس ما خارج اند. به عنوان نمونه، اخیراً شرکت آی - بی - ام<sup>۳</sup> رایانه موازی بلوجنی<sup>۴</sup> را که دارای یک میلیون پردازشگر پروتینی است در دست ساخت دارد. از طرفی می‌توان تعدادی از رایانه‌های شخصی را از طریق یک شبکه قیاس پذیر<sup>۵</sup> به هم وصل نمود و از آن به عنوان یک رایانه موازی استفاده کرد [۱]. این دو نوع رایانه موازی تحت نرم افزارهایی نظیر ام - پی - آی<sup>۶</sup> و پی - و - ام<sup>۷</sup> قابل برنامه نویسی اند. ولی مشکل برنامه نویسی با ام - پی - آی آن است که برنامه نویس با جزئیات موازی سازی از قبیل نحوه توزیع داده‌ها بر روی پردازشگرها<sup>۸</sup>، تکنیکهای موازنه کردن میزان پردازش بر روی هر یک از پردازشگرها<sup>۹</sup>، هماهنگی بین پردازشها<sup>۱۰</sup> و ارتباط بین پردازشها<sup>۱۱</sup> درگیر می‌شود [۲] که برنامه نویسی را مشکل می‌کنند.

در این تحقیق، برای موازی سازی الگوریتمهای ژنتیکی مجموعه‌ای از اسکلتها طراحی شده است. برنامه نویس با استفاده از رابط کاربر طراحی شده برای اسکلتها مختلف، می‌تواند به سادگی از آنها استفاده کند بدون آنکه از جزئیات موازی سازی مطلع شود. ضمناً برای افزایش کارایی می‌توان اسکلتها را با هم ترکیب کرد.

در این مقاله، ابتدا مقدمه‌ای بر اسکلت‌های الگوریتمی آورده شده است. سپس در بخش سوم عملکرد یک الگوریتم ژنتیکی ساده مورد بحث قرار گرفته است. در بخش چهارم به انواع روشهای موازی سازی الگوریتمهای ژنتیکی به همراه تحقیقاتی که تا به حال در این زمینه انجام شده است، اشاره می‌شود. در بخش پنجم الگوهای پیشنهادی برای موازی سازی الگوریتمهای ژنتیکی به تفصیل بحث شده است. سپس در بخش ششم،

### ۳- الگوریتمهای ژنتیکی

زمان لازم برای اجرای تابع ارزیابی، یک الگوریتم ژنتیکی ممکن است ساعتها، روزها یا حتی ماهها برای پیدا کردن یک راه حل قابل قبول طول بکشد. بنابراین این الگوریتمها با توجه به طبیعت موازی جستجوی ژنتیکی و زمان اجرای طولانی، خصوصاً در حل مسائل پیچیده کاربردی، کاندیدای بسیار خوبی برای موازی‌سازی خواهند بود. یک الگوریتم ژنتیکی موازی علاوه بر دارا بودن مزایای یک الگوریتم ژنتیکی ساده از آن سریعتر است. همچنین کمتر مستعد پیدا کردن راه‌حلهای شبه بهینه است و نیز می‌توان در طول اجرای آن از سایر تکنیکهای جستجو نیز به طور موازی بهره گرفت [۱۱ و ۱۰]. در بخشهای بعد شئوهای متفاوت موازی‌سازی الگوریتم‌های ژنتیکی به همراه خلاصه‌ای از تحقیقات انجام شده آورده شده است.

#### ۴-۱- الگوریتم ژنتیکی موازی ارباب- بنده<sup>۳۱</sup>

این الگوریتم شامل یک پردازش ارباب و تعدادی پردازش بنده است، شکل (۲-الف). پردازش ارباب الگوریتم ژنتیکی سری را اجرا می‌کند و ضمن اجرای آن، تابع ارزیابی و یا در برخی موارد عملگرهای تقاطع و جهش را به پردازشهای بنده واگذار کرده و سپس نتایج را از آنها دریافت می‌کند [۱۱ و ۱۲].

#### ۴-۲- الگوریتم ژنتیکی موازی جزیره ای<sup>۳۲</sup>

این الگوریتم شامل تعدادی پردازش است که هر کدام یک الگوریتم ژنتیکی سری را به طور مستقل اجرا می‌کند. این پردازشها را جزیره می‌نامند، شکل (۲-ب). این جزایر در فواصل زمانی مناسب، با هم تبادل اطلاعات می‌نمایند که این تبادل اطلاعات مهاجرت<sup>۳۳</sup> نامیده می‌شود. در طی فرایند مهاجرت، تعداد نسبتاً کمی از کروموزومهای هر جزیره در فواصل زمانی مشخص<sup>۳۴</sup> بر اساس توپولوژی خاصی به جزیره دیگر ارسال می‌شوند که این کروموزومها را، کروموزومهای مهاجر نامند [۱۱ و ۱۲]. کارهای بسیار زیادی در زمینه انتخاب بهترین توپولوژی مهاجرت [۱۳ - ۱۶] انجام شده است، اما به نظر می‌رسد توپولوژی ابر مکعب<sup>۳۵</sup> بهترین توپولوژی باشد.

الگوریتمهای ژنتیکی از تئوری تکامل داروین الهام گرفته‌اند. در واقع این الگوریتمها یک مدل محاسباتی از پدیده تکامل بیولوژیکی‌اند، که اولین بار در اوایل دهه هفتاد توسط جان هلند معرفی شدند [۷]. این الگوریتمها را می‌توان برای حل مسائل بهینه‌سازی یا مدل‌سازی سیستمهای تکاملی، به کار برد [۸ و ۹].

یک الگوریتم ژنتیکی ساده مطابق شکل (۱) شامل مراحل زیر است: ابتدا پارامترهای مسئله به یک رشته رمزی تبدیل می‌شوند که به این رشته اصطلاحاً "کروموزوم" می‌گویند (یک کروموزوم از مجموعه‌ای از ژنها تشکیل شده است). در واقع هر کروموزوم نسخه‌ی به رمز درآورده شده‌ای از راه‌حلهای مسئله است. سپس جمعیتی از کروموزومها به طور تصادفی ایجاد می‌شود و این جمعیت به صورت تدریجی از طریق چهار عملگر زیر تکامل پیدا می‌کند.

• ارزیابی<sup>۳۱</sup>: این عملگر از یک تابع ارزیابی برای تعیین برازندگی<sup>۳۲</sup> هر کروموزوم نسبت به سایر کروموزومها استفاده می‌کند. در واقع این عملگر میزان مناسب بودن کروموزوم را برای حل مسئله نشان می‌دهد.

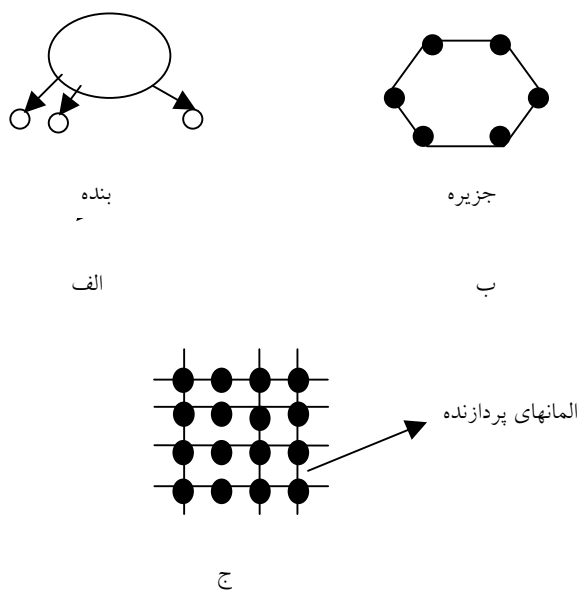
• انتخاب<sup>۳۳</sup>: این عملگر برازنده ترین کروموزومها را از جمعیت برای نسل بعدی انتخاب می‌کند.

تقاطع<sup>۳۴</sup>: این عملگر ژنهای تصادفی از کروموزوم های تصادفی را با هم تعویض می‌کند. این عملگر بر اساس احتمال مشخصی بر روی جمعیت اعمال می‌شود که این احتمال معمولاً بالاست.

• جهش<sup>۳۵</sup>: این عملگر ژنهای تصادفی از کروموزومهای تصادفی را تغییر می‌دهد. احتمال وقوع آن بر روی جمعیت کم بوده و هدف از آن حفظ تنوع ژنتیکی جمعیت در جهت میل به سمت راه حل بهینه است.

#### ۴- الگوریتمهای ژنتیکی موازی

در الگوریتمهای ژنتیکی برای حل یک مسئله، تابع ارزیابی ممکن است صدها یا هزاران بار اجرا شود، بنابراین بسته به

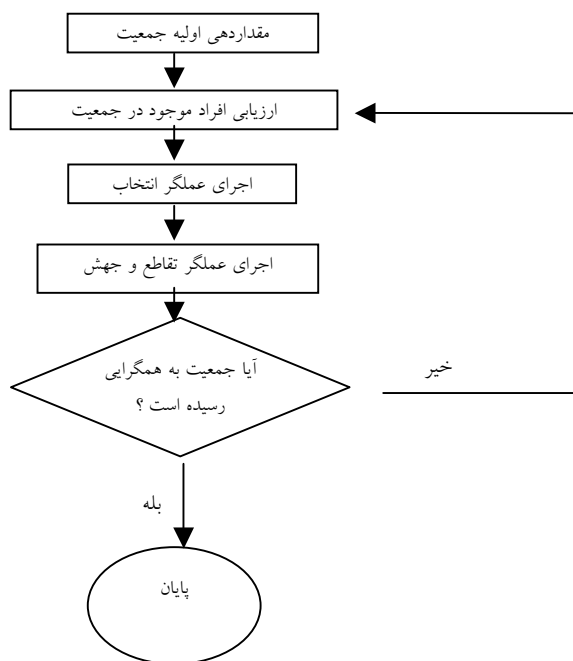


شکل ۲- الگوی شماتیک الگوریتمهای ژنتیکی موازی

الگوریتم ژنتیکی موازی جزیره‌ای قرار دارد و هر یک از گره‌ها به‌طور مستقل یکی از الگوریتمهای ژنتیکی موازی ارباب - بنده، جزیره ای یا دانه ریز را به‌طور مستقل اجرا می‌کند [۱۱ و ۱۲]. در حل مسائل پیچیده، این الگوریتم معمولاً نتایج بهتری را نسبت به سایر الگوریتمهای ژنتیکی موازی به‌دست می‌دهد. جدول (۱) برخی از تحقیقات انجام شده در زمینه الگوریتمهای ژنتیکی موازی را به‌طور خلاصه نشان می‌دهد.

### ۵- الگوهای پیشنهادی برای الگوریتمهای ژنتیکی موازی

اغلب تحقیقات اولیه انجام شده در زمینه اسکلت‌های الگوریتمی، بر ارائه ساختارهای متداول برنامه نویسی موازی متمرکز شده‌اند که از آن جمله به الگوهای تقسیم و حل، پردازش لوله‌ای<sup>۳۴</sup> و نگاهشت [۶، ۴، ۳] می‌توان اشاره کرد. در حالی که تحقیقات محدودی در زمینه‌های کاربردی نظیر پردازش تصویر، مدل‌سازی جامدات و غیره انجام شده است. از این رو در این مقاله با توجه به محبوبیت الگوریتمهای ژنتیکی



شکل ۱- نمودار اجرایی یک الگوریتم ژنتیکی ساده

### ۴-۳- الگوریتم ژنتیکی موازی دانه ریز<sup>۳۱</sup>

معمولاً این الگوریتم بر روی ماشینهای موازی بزرگ با معماری توری<sup>۳۲</sup> دوبعدی پیاده‌سازی می‌شود، شکل (۲- ج). در این الگوریتم یک یا چند کروموزوم از جمعیت بر روی هر پردازشگر قرار گرفته و سپس هر پردازشگر یک الگوریتم ژنتیکی سری را اجرا می‌کند، اما در اجرای الگوریتم ضمن اجرای عملگر انتخاب، بهترین کروموزومها را از میان همسایگان خود (شمال - جنوب - شرق - غرب) انتخاب کرده و سپس عملگرهای تقاطع و جهش را بر روی جمعیت برگزیده اجرا می‌کند. کروموزومهای فرزند در صورتی که بهتر از کروموزومهای موجود در پردازشگر باشند جایگزین آنها می‌شوند [۱۱ و ۱۲]. در این الگوریتم به دلیل همپوشانی بین پردازشگرهای همسایه، امکان گسترش راه‌حل‌های خوب در کل جمعیت وجود دارد.

### ۴-۴- الگوریتم ژنتیکی موازی سلسله مراتبی<sup>۳۳</sup>

این الگوریتم از دو الگوریتم ژنتیکی موازی به‌صورت سلسله مراتبی تشکیل می‌شود. در این الگوریتم در بالاترین سطح، یک

جدول ۱- چند پیاده سازی از الگوریتمهای ژنتیکی موازی

الگوریتم ژنتیکی موازی	مرجع	نوع موازی سازی	توپولوژی	کاربرد
ASPARAGOS	[۱۷]	الگوریتم دانه ریز	نردبانی*	فروشنده دوره گرد
CoPDEB	[۱۸]	الگوریتم جزیره ای	کاملاً متصل	بهینه سازی توابع
DGENESIS 1.0	[۱۶]	الگوریتم جزیره ای	دلخواه	بهینه سازی توابع
ECO-GA	[۱۹]	الگوریتم دانه ریز	توری	بهینه سازی توابع
EnGENEer	[۲۰]	الگوریتم ارباب - بنده	ارباب - بنده	متنوع
GALOPPS 3.1	[۲۱]	الگوریتم جزیره ای	دلخواه	بهینه سازی توابع و حل مسئله حمل و نقل
GAMES	[۲۲]	الگوریتم جزیره ای	-----	بهینه سازی توابع و ...
GDGA	[۲۳]	الگوریتم جزیره ای	----	بهینه سازی توابع
GENITOR II	[۲۴]	الگوریتم جزیره ای	حلقه	بهینه سازی توابع
HSDGA	[۲۵]	الگوریتمهای جزیره ای - دانه ریز- سلسله مراتبی	حلقه ، درخت ، ستاره** و ...	بهینه سازی توابع
PeGAsuS	[۲۶]	الگوریتمهای جزیره ای - دانه ریز	دلخواه	بهینه سازی توابع.
PGAPack	[۲۷]	الگوریتم ارباب - بنده	ارباب - بنده	بهینه سازی توابع
RPL2	[۲۸]	الگوریتمهای ارباب - بنده و دانه ریز	دلخواه	مسائل تحقیقی و تجاری
SGA-Cube	[۲۹]	الگوریتم ارباب - بنده از ماشین ان-کیوب ۲*** استفاده شده است.	ابر مکعب	بهینه سازی توابع

\*Lader                      \*\*star                      \*\*\*nCUBE2

شده می‌توان یک مدل کارایی به‌دست آورد. در نتیجه کارایی اسکلتها قابل پیشگویی است. این اسکلتها کارایی خوبی دارند درحالی‌که در تحقیقات دیگر [۵ و ۳۰ - ۳۲] چون اسکلتها به برنامه نویسی تابعی پیاده‌سازی شده‌اند پیشگویی کارایی آنها میسر نیست. ضمناً کارایی پایینی دارند زیرا کامپایلرهای تابعی عمل نگاشت پردازشها را نمی‌توانند به بهترین وجه انجام دهند. در این الگوها پردازشهای موازی از طریق تبادل پیام با هم ارتباط برقرار می‌کنند. اما برنامه نویس در استفاده از آنها مستقیماً درگیر توابع مربوط به تبادل پیامها نمی‌شود، بلکه این توابع در کد مربوط به پیاده سازی الگوها گنجانده شده است. بنابراین مطابق شکل (۳) این کتابخانه به‌صورت لایه‌ای بین دستورات سطح پایین برنامه نویسی موازی و برنامه کاربر قرار می‌گیرد. در این پژوهش، خوشه<sup>۳۵</sup> ای از رایانه‌های شخصی از

در میان الگوریتمهای بهینه سازی، مجموعه‌ای از الگوها برای موازی‌سازی الگوریتمهای ژنتیکی به‌صورت کتابخانه‌ای از کلاسها طراحی شده است. با استفاده از الگوهای طراحی شده، برنامه نویس می‌تواند از محیط برنامه نویسی ویژوال ++C به‌عنوان یک محیط برنامه‌نویسی موازی استفاده کند و با فراخوانی شیء مربوطه، یک الگوی موازی برای برنامه خود انتخاب کرده و آن را با توابع مربوط به مسئله اش پر کند، در این صورت لازم نیست از زیر ساخت الگوی انتخابی و مسائل مربوط به اجرای موازی آن مطلع باشد. البته برنامه نویس می‌تواند با تعیین توپولوژی مجازی پردازشها در تعیین ارتباط منطقی پردازشها و در نتیجه در زمان اجرای اسکلت نقش داشته باشد که در سایر تحقیقات در زمینه اسکلتهای الگوریتمی نظیر کار دارلینگتون [۴] و پلاگاتی [۵] وجود ندارد. همچنین برای هر یک از اسکلتها طراحی

برنامه کاربر
کتابخانه الگوهای موازی
لایه انتقال پیام (ام-پی-آی - سی-اچ)

شکل ۳- معماری لایه ای یک برنامه کاربردی با استفاده از کتابخانه الگوهای پیشنهادی

طریق پرتکل اینترنت<sup>۳۶</sup> به هم متصل شده و به عنوان ماشین موازی استفاده شده است. چنانچه این شبکه ارتباطی قیاس پذیر باشد این نوع ماشین موازی از کارایی مناسبی برخوردار است. این نوع ماشین به علت سادگی و نیز ارزانی اخیراً مورد توجه قرار گرفته است. نصب یک مدل انتقال پیام استاندارد از قبیل ام-پی-آی یا پی-و-ام این شبکه را به صورت یک ماشین موازی واحد در اختیار کاربر می گذارد. الگوهای انتخاب شده بر روی ماشین موازی فوق پیاده سازی شده است و نتایج عملی به دست آمده، نتایج واقعی است و هیچ گونه شبیه سازی در این پژوهش انجام نشده است. برخی الگوها ترکیبی از الگوهای دیگر هستند که دستاورد جدیدی در تحقیقات انجام شده در زمینه اسکلت های الگوریتمی است و در بسیاری از تحقیقات انجام شده در این زمینه از جمله سیستم های ALWAN [۳۳]، HDC [۳۴] انجام نشده است. قبل از معرفی الگوها به اختصار کتابخانه های استفاده شده در این پژوهش آورده شده است.

#### الف- توضیحی بر کتابخانه های استفاده شده

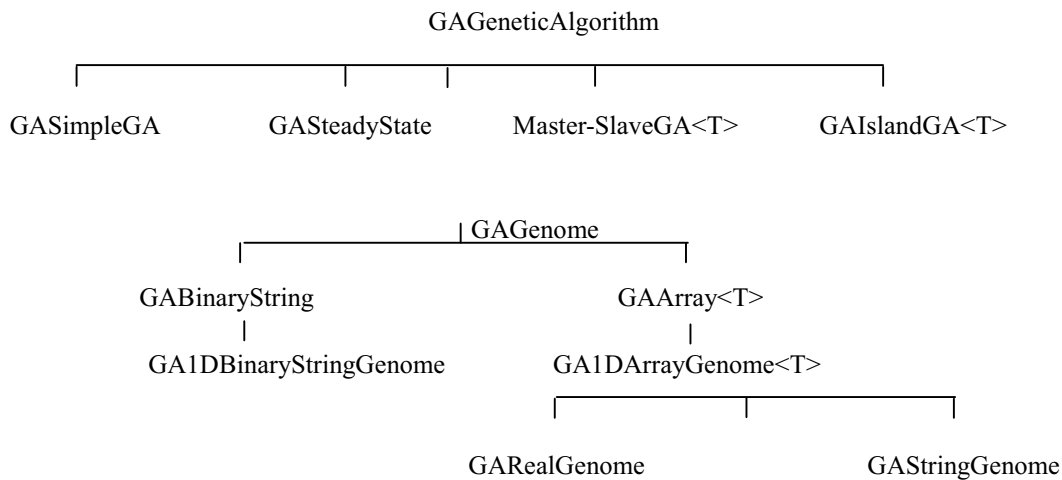
کتابخانه الگوهای پیشنهادی، توسعه یافته کتابخانه GALib نسخه ۲.۴.۴ [۳۵] است. این کتابخانه در دانشگاه ماساچوست طراحی شده و شامل مجموعه ای از اثبات به زبان C++ برای طراحی الگوریتم های ژنتیکی سری است و در آن ابزارهایی برای استفاده از الگوریتم های ژنتیکی در حل مسائل بهینه سازی در هر برنامه C++ ای فراهم شده است.

قبل از معرفی مجموعه الگوهای پیشنهادی، ابتدا سلسله مراتب کلاسها را شکل (۴) در آنها به اختصار شرح می دهیم. در

این مجموعه دو شیء پایه به نامهای GAGeneticAlgorithm و GAGenome داریم. GAGeneticAlgorithm یک کلاس مجرد است که تمام الگوهای پیشنهادی از آن ارث بری می کنند. این شیء سیر تکاملی الگوریتم و نیز مجموعه ای از پارامترها را نظیر نرخ جهش، نرخ تقاطع و معیار خاتمه الگوریتم و... را مشخص می کند. دو شیء GASimpleGA و GASteadyState از کلاس GAGeneticAlgorithm ارث بری می کنند و برای پیاده سازی یک الگوریتم ژنتیکی سری می توان از آنها استفاده کرد. تفاوت این دو در آن است که در شیء GASteadyState در هر نسل جمعیت جدید به جمعیت قبلی اضافه شده و سپس از جمعیت حاصل بدترین افراد حذف می شوند تا اندازه جمعیت ثابت بماند در حالی که در شیء GASimpleGA در هر نسل، جمعیت جدید جایگزین جمعیت قبلی می شود. از دو شیء GAIslandGA، Master-SlaveGA برای پیاده سازی یک الگوریتم ژنتیکی موازی می توان استفاده کرد. این دو شیء به صورت قالب<sup>۳۷</sup> طراحی شده اند. به این مفهوم که می توان هر یک از کلاسهای GASimpleGA و GASteadyState را به عنوان الگوریتم ژنتیکی سری به درون آنها ارسال کرد. البته در کلاس GAIslandGA می توان خود این کلاس و یا کلاس Master-SlaveGA را نیز درون آن ارسال کرد و یک الگوریتم ژنتیکی سلسله مراتبی ساخت.

برای استفاده از یک الگوریتم ژنتیکی سری باید بتوان راه حل مسئله را به صورت ساختمان داده ای نمونه ای نمایش داد تا بر اساس آن جمعیتی از راه حلها تولید شود. این ساختمان داده نمونه توسط شیء GAGenome تعریف می شود و به آن کروموزوم نیز می گویند. این کتابخانه شامل دو نوع کروموزوم GABinaryStringGenome و GAArrayGenome است. این کروموزومها از کلاس پایه GAGenome و کلاس مربوط به ساختمان داده شان ارث بری می کنند. به عنوان مثال کلاس GABinaryStringGenome از دو شیء GABinaryString و GABinaryStringGenome مشتق می شود.

شیء GABinaryString یک پیاده سازی ساده از یک رشته



شکل ۴- سلسله مراتب کلاسها

است که در دانشگاه میسیسیپی طراحی شده است و می توان آن را بر روی شبکه ای از ایستگاههای کاری اجرا کرد و از آن شبکه به عنوان یک ماشن موازی واحد استفاده کرد. این کتابخانه اجازه می دهد پردازشهای روی ماشنهای متفاوت از طریق پرتکل TCP/IP و یا حافظه اشتراکی با هم ارتباط برقرار کنند. این نسخه بر روی سیستم عامل ویندوز 2000 / NT کارایی بسیار خوبی دارد و محیطی را برای اجرا از راه دور به طور کامل فراهم می کند. در بخش زیر الگوهای پیشنهادی به تفصیل مورد بحث قرار می گیرند.

#### ب- SingletonGA

این اسکلت، یک الگوی تک پردازشی است که یک الگوریتم ژنتیکی سری را اجرا می کند، شکل (۵). دو نوع الگوریتم ژنتیکی سری در این کتابخانه تعبیه شده است: الگوریتم ژنتیکی ساده<sup>۳۹</sup> و الگوریتم ژنتیکی حالت پایدار<sup>۴۰</sup> که به ترتیب با دو کلاس GASimpleGA و GASteadyStateGA مشخص شده و به عنوان هسته مرکزی سایر الگوها محسوب می شوند. واسط برنامه نویسی این الگو به صورت زیر است:

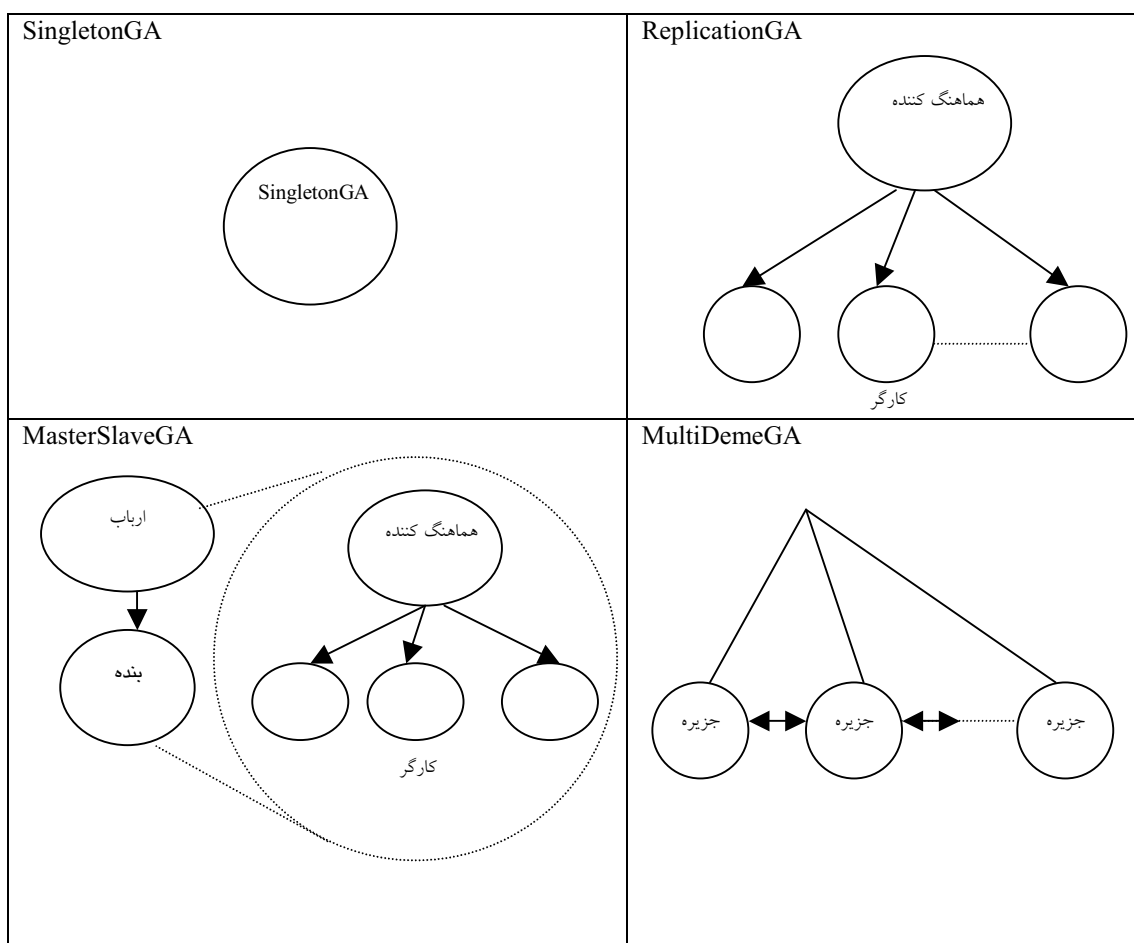
```

GASimpleGA ( const GAGenome & g )
GASteadyStateGA ( const GAGenome & g )
  
```

بیتی است. شیء GAIDBinaryStringGenome کروموزومی را به صورت یک رشته دودویی تعریف می کند این رشته دودویی ممکن است طول ثابت یا متغیر داشته باشد.

شیء GAArray<T> یک پیاده سازی از یک آرایه است و به صورت قالب پیاده سازی شده است. هر یک از عضوهای این آرایه می تواند از هر نوعی باشد. شیء GAIDArrayGenome<T> کروموزومی را به صورت یک آرایه یک بعدی تعریف می کند و از دو کلاس GAArray<T> و GAGenome مشتق می شود. هر عضو آرایه یک ژن است که نوع آن توسط T در این قالب مشخص می شود. به عنوان مثال اگر T از نوع صحیح باشد کروموزومی به صورت رشته ای از اعداد صحیح تعریف می شود. می توان با استفاده از این کلاس کروموزومهای اعشاری و کاراکتری نیز ساخت اما به دلیل کثرت استفاده از آنها دو کلاس مجزا به صورت دو شیء GARealGenome و GASTringGenome در نظر گرفته شده است.

در این تحقیق برای ایجاد پردازشهای موازی از کتابخانه ام پی - پی - سی - اچ<sup>۳۸</sup> نسخه ۲.۰.۱ [۳۶] استفاده شده است. این کتابخانه نسخه ای از مدل انتقال پیام ام پی - پی - آ



شکل ۵- اسکلت های پیشنهادی

یکنواخت<sup>۴۳</sup> است، تعیین کرد. علاوه بر آن نوع عملگر انتخاب را که شامل چرخ رولت<sup>۴۴</sup>، تورنومنت و غیره است، نیز می توان در الگوریتم ژنتیکی خود تعیین کرد.

### ج - ReplicationGA

این الگو شامل یک هماهنگ کننده<sup>۴۵</sup> و مجموعه ای از کارگرها<sup>۴۶</sup> است (شکل ۵). پردازش هماهنگ کننده، جمعیتی از کروموزومها را از ورودی می گیرد و بسته به تعداد کارگرها، زیر مجموعه ای از جمعیت ورودی را به هر یک از کارگرها می فرستد. کارگرها به محض دریافت زیر جمعیت، شروع به اجرای عملگر ژنتیکی می کنند. این عملگر ممکن است یکی از توابع ارزیابی، تقاطع یا جهش باشد که به عنوان پارامتر به این الگو ارسال شده است، سپس نتیجه را به هماهنگ کننده

پارامتر ورودی، متغییری از کلاس GAGenome است. این کلاس، یک کلاس پایه برای تعیین نوع کروموزوم است و چهار کلاس GAIDBinaryStringGenome, GARealGenome, GAStringGenome, GAIDArrayGenome < int > از آن ارث بری می کنند که به ترتیب چهار نوع کروموزوم دودویی، اعشاری، کاراکتری و عددی را مشخص می کنند. بنابراین می توان برای هر کاربردی کروموزوم مناسب با آن را انتخاب کرد.

پارامترهایی نظیر سرعت تقاطع، سرعت جهش، اندازه جمعیت و تعداد نسلها بافراخوانی توابع موجود در این کلاس مقدار دهی می شوند. همچنین با استفاده از این کتابخانه می توان نوع عملگر تقاطع را که شامل تک نقطه ای<sup>۴۱</sup>، دونقطه ای<sup>۴۲</sup> و



برمی‌گردانند. پردازش هماهنگ کننده پس از دریافت نتایج از تمام کارگرها، آنها را با هم ترکیب کرده و به‌عنوان خروجی الگو برمی‌گرداند. واسط برنامه نویسی این الگو به‌صورت زیر است:

```
ReplicationGA(GAPopulation& pop ,int nprocess , char operation , char topology)
```

• پارامتر اول متغیری از کلاس GAPopulation است. این متغیر جمعیتی از کروموزومها را مشخص می‌کند که باید بر روی آنها، یکی از عملگرهای ژنتیکی (ارزیابی، تقاطع، جهش) توسط این الگو اجرا شود.

• پارامتر دوم (nprocess)، تعداد کارگرها در این الگو را مشخص می‌کند.

• پارامتر سوم (operation)، مشخص می‌کند که کدام یک از عملگرهای ژنتیکی باید توسط کارگرها بر روی جمعیت ورودی اجرا شود. این پارامتر سه کاراکتر E و C و M را به‌ترتیب برای سه عملگر ارزیابی، تقاطع و جهش می‌گیرد.

• پارامتر چهارم (topology)، توپولوژی مجازی پردازشها را در این الگو مشخص می‌کند. توپولوژی مجازی می‌تواند به صورت ابر مکعب، توری، حلقه<sup>۴۷</sup> و یا به‌صورت پیش فرض درخت باشد.

تعیین توپولوژی مجازی یکی از نقاط قوت این الگو محسوب می‌شود، زیرا در اغلب اسکلت‌های معرفی شده تا به حال برنامه‌نویس هیچ نقشی در زمان اجرای اسکلت نداشته است، درحالی‌که در این الگو کاربر می‌تواند با انتخاب توپولوژی مجازی پردازشها در زمان اجرای آن نقش داشته باشد.

#### د- Master-SlaveGA

این الگو مرکب از دو الگوی ReplicationGA و SingletonGA است، شکل (۵). در این الگو پردازش ارباب در واقع یک SingletonGA ناقص است درحالی‌که بنده یک ReplicationGA است. پردازش ارباب ابتدا جمعیتی از کروموزومها را به‌صورت تصادفی تولید کرده سپس یک الگوریتم ژنتیکی سری را اجرا می‌کند اما در اجرای الگوریتم

ژنتیکی، اجرای برخی عملگرها را به پردازش بنده که در واقع نمونه‌ای از الگوی ReplicationGA است واگذار می‌کند. این عملگرها توسط پارامترهای ورودی الگو مشخص می‌شوند. واسط برنامه نویسی این الگو به‌صورت زیر است:

```
MasterSlaveGA(const GAGenome& g ,int nprocess ,char opr1,opr2 ,char topology )
```

• پارامتر اول متغیری از کلاس GAGenome است. این پارامتر، نوع کروموزوم را در الگوی SingletonGA درون این الگو مشخص می‌کند.

• پارامتر دوم (nprocess)، تعداد کارگرها را در الگوی ReplicationGA درون این الگو مشخص می‌کند.

• پارامتر سوم و چهارم (opr1,opr2) عملگرهای ژنتیکی ای را که باید توسط الگوی ReplicationGA درون این الگو به‌طور موازی اجرا شوند، مشخص می‌کند. حداکثر دو عملگر از بین عملگرهای ژنتیکی (ارزیابی، تقاطع، جهش) را می‌توان در این الگو به بنده واگذار کرد.

• پارامتر چهارم (topology) توپولوژی مجازی پردازشها را در این الگو مشخص می‌کند و مانند پارامتر مشابه‌اش در الگوی ReplicationGA مقدار می‌گیرد

از جمله نقاط قوت این الگو می‌توان به ترکیب دو الگوی SingletonGA و ReplicationGA اشاره کرد. از دیگر نقاط قوت این الگو تعیین توپولوژی مجازی پردازشها توسط برنامه نویس و در نتیجه نقش داشتن او در زمان اجرای الگوست.

#### ه- Multi-DemeGA

این الگو شامل تعدادی الگوهای کوچک SingletonGA است که به هر یک، زیر جمعیت<sup>۴۸</sup> یا جزیره اطلاق می‌شود، شکل (۵). زیر جمعیتها در شرایط خاصی با هم ارتباط برقرار می‌کنند که این ارتباط، شامل ارسال تعدادی کروموزوم از یک زیر جمعیت به دیگری بوده که مهاجرت نامیده می‌شود. در این الگو دو نوع مهاجرت (همگام<sup>۴۹</sup> و غیر همگام<sup>۵۰</sup>) در نظر گرفته شده است که در نوع غیر همگام، مهاجرت بین زیر جمعیتها پس از آنکه هر یک از زیر جمعیتها کاملاً همگرا شدند، اتفاق می‌افتد [۳۷، ۱۲]. درحالی‌که در نوع همگام مهاجرت هر چند

نسل، یکبار اتفاق می افتد [۳۸]. در این اسکلت الگوی انتخاب شده برای هر مهاجرت به صورت زیر است:

با فرض اینکه زیر جمعیتها به طور مجازی و به صورت حلقه مرتب شده اند، در اولین مهاجرت، هر زیر جمعیت، کروموزومهای مهاجرش را به اولین همسایه سمت چپش می فرستد. در دومین مهاجرت، هر زیر جمعیت، کروموزومهای مهاجرش را به دومین همسایه سمت چپش می فرستد و این عمل تکرار می شود تا یک زیر جمعیت طی مهاجرتهای متعدد، کروموزومهای مهاجر خود را به سایر زیر جمعیتها (بجز خودش) ارسال کند. سپس این چرخه مجدداً تکرار می شود. در این الگو، کروموزومهای مهاجر از بهترین کروموزومهای زیر جمعیت انتخاب شده و جایگزین بدترین کروموزومهای زیر جمعیت مقصد می شوند. واسط برنامه نویسی این الگو به صورت زیر است:

```
GAIslandGA(const GAGenome& g , int nprocess , int migtype , char topology )
```

- پارامتر اول متغیری از کلاس GAGenome است. این پارامتر، نوع کروموزوم را در الگوی SingletonGA درون این الگو مشخص می کند.
- پارامتر دوم (nprocess) ، تعداد زیر جمعیتها (جزایر) را در این الگو مشخص می کند.
- پارامتر سوم (migtype) ، نوع مهاجرت در الگو را مشخص می کند که ممکن است همگام یا غیر همگام باشد.
- پارامتر چهارم (topology) ، توپولوژی مجازی پردازشها را مشابه سایر الگوها مشخص می کند.

از جمله نقاط قوت این الگو، می توان به ترکیب الگوهای کوچک SingletonGA و تعیین توپولوژی مجازی پردازشها توسط برنامه نویس اشاره کرد. پارامترهایی نظیر تعداد نسلها بین دو مهاجرت و تعداد مهاجرین از طریق فراخوانی روالهای موجود در این الگو، قابل مقاداردهی هستند.

### و- تکنیک به رمز درآوری

جمعیت در هر الگو آرایه ای از اشاره گرهای کلاس GAGenome است، لذا برای تبادل جمعیت یا بخشی از آن بین

پردازشگرها، با آرایه ای از اشاره گرها سروکار داریم. اما از آنجا که مبادله اشاره گرها سبب ایجاد نتایج نادرست می شود، لذا باید این آرایه یا بخشی از آن قبل از ارسال به صورت رشته ای به رمز درآورده شود. این عمل با ارسال جمعیت به تابع Encode از کلاس CommunicateGenome انجام می گیرد. برای رمزگشایی نیز تابع Decode در این کلاس در نظر گرفته شده است. لازم به توضیح است که فراخوانی این توابع در هر الگو به طور خودکار قبل از هر تبادل اطلاعات انجام می گیرد.

### ۵-۱ مثال

برای تولید یک برنامه موازی با استفاده از این کتابخانه باید مراحل زیر طی شود:

الف- انتخاب کروموزوم مناسب با توجه به نوع مسئله

ب- انتخاب الگوی مناسب از کتابخانه

ج- مقاداردهی الگو با توجه به مسئله

د- اجرای الگوی انتخابی

در برنامه زیر، می خواهیم با استفاده از الگوی GAIslandGA کروموزوم دودویی را پیدا نماییم که تعداد یکهای آن ماکزیمم باشد. برای این منظور تابع OneMax به عنوان تابع برازندگی تعریف شده است. این تابع بر اساس تعداد یکهای موجود در کروموزوم یک مقدار برازندگی به آن نسبت می دهد. (توضیح: در تابع OneMax، توابع genome.length(), genome.gene() به ترتیب مقدار ژن i ام در کروموزوم و طول کروموزوم را برمی گردانند.) در برنامه ابتدا یک کروموزوم دودویی از کلاس GAIDBinaryStringGenome بنام genome گرفته شده است. طول کروموزوم ۳۲ بیت و تابع OneMax نیز برای تعیین مقدار برازندگی آن در نظر گرفته شده است، خط (۱). سپس الگوی GAIslandGA از کتابخانه الگوها انتخاب شده است، خط (۲). با توجه به نحوه پیاده سازی این الگو به صورت قالب کلاس GASimpleGA به عنوان الگوریتم ژنتیکی سری به درون آن ارسال شده است. سپس با توجه به واسط برنامه نویسی این الگو

زمان اجرای این الگو شامل دو بخش می‌باشد:  $T_{comp}$  که در واقع زمان صرف شده برای اجرای عملگرژنتیکی، توسط هریک از کارگراهاست و  $T_{comm}$  که شامل زمان لازم برای ارسال بخشی از جمعیت از هماهنگ کننده به هر یک از کارگراها و دریافت نتایج از آنهاست. برای محاسبه  $T_{comp}$  فرض شده است که فقط تابع ارزیابی به کارگراها واگذار می‌شود و از دو عملگر تقاطع و جهش به دلیل سادگی و کوتاه بودن زمان اجرای شان در مقابل تابع ارزیابی صرف نظر شده است.

اگر  $n$  تعداد کل کروموزومهای موجود در جمعیت،  $T_f$  زمان لازم برای اجرای تابع ارزیابی بر روی یک کروموزوم،  $p$  تعداد پردازشگرها و در نتیجه  $p-1$  تعداد کارگراها باشد در آن صورت:

$$T_{comp} = \frac{nT_f}{p-1} \quad \text{صورت:}$$

برای محاسبه  $T_{comm}$  چنانچه ارسال هر زیرجمعیت از هماهنگ کننده به هر یک از کارگراها  $T_c$  واحد زمانی طول بکشد، در آن صورت ارسال تمام زیر جمعیتها  $(p-1)T_c$  واحد زمانی طول خواهدکشید. پس از ارسال زیر جمعیتها مطابق شکل (۶)، هماهنگ کننده قبل از اتمام کار آخرین کارگر، نتایج کارگرهای قبلی را جمع آوری می‌کند سپس  $T_c$  واحد زمانی طول می‌کشد تا نتایج آخرین کارگر را دریافت کند. پس

$$T_{comm} = (p-1)T_c + T_c = pT_c$$

$$T_p = T_{comp} + T_{comm} = \frac{nT_f}{p-1} + pT_c \quad (۱)$$

چنانچه از  $T_p$  نسبت به  $p$  مشتق بگیریم می‌توانیم تعداد بهینه کارگرها را به دست آوریم.

$$S^* = p-1 = \sqrt{\frac{nT_f}{T_c}} \quad (۲)$$

در این الگو  $T_c$  را می‌توان به صورت رابطه خطی  $T_c = Bx + L$  نمایش داد که در آن  $x$  حجم اطلاعات،  $B$  عکس پهنای باند و  $L$  زمان لازم برای افزودن سرآمد<sup>۱</sup> به هر پیام است و بستگی به سیستم عامل، محیط برنامه نویسی و سخت افزار دارد. لذا زمان لازم برای ارسال کروموزومها به هر کارگر

که اشاره شد، پارامتر genome به عنوان نوع کروموزوم، تعداد زیر جمعیتها برابر پنج، الگوی مهاجرت همگام و توپولوژی مجازی پردازشها نیز حلقه انتخاب شده است. در خط (۳) با فراخوانی تابع  $nInterval()$  تعداد نسلها بین دو مهاجرت متوالی برابر پنج و در خط (۴) نیز با فراخوانی تابع  $nMigration()$  تعداد مهاجرین ده کروموزوم تعیین شده است. سایر پارامترها نظیر تعداد نسلها، اندازه جمعیت، احتمال تقاطع و احتمال جهش از طریق فراخوانی سایر توابع، قابل مقدار دهی هستند. در خط (۵) پارامترهای موجود در این الگو مقداردهی اولیه شده و در خط (۶)، اسکلت اجرا می‌شود.

1. GA1DBinaryStringGenome genome(32,OneMax);
2. GAIslandGA<GASimpleGA> ga (genome,5, synchronous,ring);
3. ga.nInterval(5);
4. ga.nMigration(10);
5. ga.initialize(seed);
6. ga.RunSkeleton();

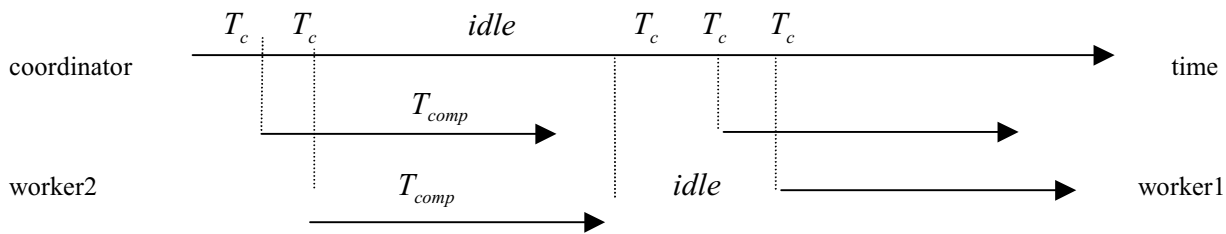
```
float OneMax( GAGenome & g)
{
double count=0;
GAID binary String Genome & genome
= ( GA1DBinaryStringGenome & ) g;
Float score=0.0;
For (int I=0 ;I<genome.length();I++)
Score+=genome.gene(I);
Score/=genome.length();
return(score);
}
```

## ۶- مدل کارایی اسکلت‌های پیشنهادی

در بخش زیر به تفکیک مدل کارایی هر یک از اسکلت‌های پیشنهادی آمده است.

### الف - ReplicationGA

برای محاسبه زمان اجرای این الگو، سه فرض زیر در نظر گرفته شده است: ۱- هر پردازش بر روی یک پردازشگر اجرا می‌شود ۲- تعداد افرادی که در هر مرحله به یک کارگر نسبت داده می‌شود ثابت است. ۳- زمان اجرای تابع ارزیابی برای تمام کروموزومهای موجود در جمعیت، ثابت فرض شده است.



شکل ۶- الگوی شماتیک از زمان اجرای الگوی ReplicationGA

انتخاب، تقاطع و جهش در مقابل زمان اجرای تابع ارزیابی می‌توان از این زمانها در مقابل  $T_f$  صرف نظر کرد بنابراین

$$T_{comp} \cong \frac{nT_f}{p-1}$$

$T_{comm}$  نیز در این الگو برابر  $T_{comm}$  در پردازش بنده و در نتیجه در الگوی ReplicationGA بوده و برابر  $pT_c$  است، بنابراین

$$T_p = \frac{nT_f}{p-1} + pT_c$$

در این الگو Speedup که برابر خارج قسمت زمان اجرای الگوریتم ژنتیکی سری، به زمان اجرای الگوریتم موازی است به صورت زیر خواهد بود:

$$Speedup = \frac{nT_f}{\frac{nT_f}{p-1} + pT_c}$$

استفاده از این الگو زمانی مفید است که تابع ارزیابی، یک تابع زمانبر باشد، در صورتی که زمان اجرای تابع ارزیابی کوتاه باشد ممکن است زمان اجرای این الگو از زمان اجرای الگوریتم سری بیشتر شود. در این الگو چنانچه  $T_f \gg T_c$  و اندازه جمعیت نیز بزرگ باشد می‌توان Speedup ای نزدیک Speedup خطی به دست آورد.

### ج- Multi-DemeGA

برای محاسبه زمان اجرای این الگو فرضهای زیر در نظر گرفته شده است: ۱- هر زیر جمعیت بر روی یک پردازشگر اجرا می‌شود. ۲- اندازه تمام زیر جمعیتها برابر و مساوی  $n/p$  است که  $n$  تعداد کل کروموزومها و  $p$  تعداد زیر جمعیتها یا به

برابر  $T_{send} = B \frac{nl_i}{p-1} + L$  و زمان لازم برای دریافت مقدار برازندگی هر زیرجمعیت  $T_{rec} = B \frac{nl_f}{p-1} + L$  خواهد بود که در آنها  $l_i$  طول هر کروموزوم و  $l_f$  طول مقدار برازندگی کروموزوم است. بنابراین زمان تخمین زده شده در هر نسل به طور دقیقتر برابر معادله زیر خواهد بود:

$$T_p = (p-1)T_{send} + T_{rec} + \frac{nT_f}{p-1}$$

### ب- Master-SlaveGA

برای محاسبه زمان اجرای این الگو فرض شده است که هر پردازش بر روی یک پردازشگر قرار دارد و فقط تابع ارزیابی در این الگو توسط بنده که نمونه‌ای از الگوی ReplicationGA است به طور موازی انجام می‌شود. ضمناً عملگرهای انتخاب، تقاطع و جهش به دلیل سادگی و کوتاه بودن زمان اجرای آنها به طور سری توسط ارباب انجام می‌شوند. بنابراین  $T_{comp}$  در این الگو برابر معادله زیر خواهد بود:

$$T_{comp} = T_{selection} + T_{crossover} + T_{mutation} + T'_{comp} = nt_{selection} + np_{crossover}t_{crossover} + np_{mutation}t_{mutation}$$

که در آن  $n$  تعداد کروموزومها،  $p_{crossover}$ ،  $p_{mutation}$  به ترتیب احتمال جهش و احتمال تقاطع و  $t_{selection}$ ،  $t_{crossover}$ ،  $t_{mutation}$  به ترتیب زمان اجرای عملگرهای انتخاب، تقاطع و جهش بوده و  $T'_{comp}$  نیز برابر  $T_{comp}$  در الگوی ReplicationGA است. در این معادله به دلیل کوچک بودن زمان اجرای عملگرهای

$$\text{Speedup} = \frac{gnT_f}{k(g_c \frac{n}{p} T_f + 2T_c)}$$

g تعداد نسلا در یک الگوریتم ژنتیکی سری است.

## ۷- آزمایشهای انجام شده

همانطور که در بخش ۵ اشاره شد در این تحقیق، شبکه‌ای از رایانه‌های شخصی به‌عنوان ماشین موازی استفاده شده است و اجرای برنامه کامپایل شده بر روی این شبکه و تمامی ارتباطات بین پردازشهای موجود بر روی پردازشگرهای متفاوت در سطح پایین از طریق نرم افزار ام - پی - آی - سی - اچ انجام گرفته است. با توجه به کندی نسبی محیط ارتباطی برای آزمون الگوهای پیشنهادی و نشان دادن مزیت استفاده از الگوریتمهای ژنتیکی موازی در حل مسائل زمانبر، استفاده از توابع آزمون ساده نظیر مثال مطرح شده در بخش (۵)، هیچ بهبودی در زمان اجرا نسبت به حالت سری ایجاد نمی‌کند. به همین دلیل در مجموعه آزمایشات انجام شده تاکید بر آن بوده است که تابع ارزیابی یک تابع زمانبر بوده و بار محاسباتی زیادی داشته باشد لذا مسئله‌ای با یک تابع ارزیابی فرضی در نظر گرفته شده است که زمان اجرای آن قابل تغییر است و یکبار زمان اجرا را ۲ میلی ثانیه و بار دیگر ۴ میلی ثانیه گرفته و الگوها را آزموده‌ایم. در آزمونهای انجام شده برای آنکه بتوانیم به لحاظ زمانی، مقایسه درستی بین این الگو و یک الگوریتم ژنتیکی سری داشته باشیم سعی شده است کیفیت راه حل به دست آمده توسط هر دو تقریباً یکسان باشد همانطور که در تحقیقات سایرین نیز این مسئله مد نظر قرار گرفته است [۱۲].

برای آزمون هر دو الگو از جمعیتی با ۱۲۰ کروموزم به طول ۸۰ استفاده شده است و زمان اجرای تابع ارزیابی بر روی هر کروموزوم یکبار ۲ میلی ثانیه و بار دیگر ۴ میلی ثانیه انتخاب شده است.

شکل (۷) نمودار زمان اجرای الگوی Master-SlaveGA را به صورت تجربی و تئوری به ازای  $T_f = 2ms$  نشان می‌دهد. در این آزمایش  $T_c \cong 27ms$  بوده و بنا به معادله (۲) تعداد

عبارت دیگر تعداد پردازشگرهاست. در این الگو چون هر زیر جمعیت یک SingletonGA است لذا  $T_{comp}$  در یک نسل برابر  $T_{comp}$  در یک نسل از یک الگوریتم ژنتیکی سری است و برابر  $T_f \frac{n}{p}$  است (البته در محاسبه زمان اجرای الگوریتم ژنتیکی سری از  $T_{crossover}$ ،  $T_{mutation}$  در مقابل  $T_f$  صرف نظر شده است).

همانطور که گفته شد، در این الگو در هر بار مهاجرت، هر یک از زیر جمعیتها به‌طور همزمان درصدی از بهترین کروموزومهای خود را به یکی از زیر جمعیتهای همسایه می‌فرستند بنابراین چنانچه  $T_c$  زمان لازم برای ارسال کروموزومها از یک زیر جمعیت به دیگری باشد در آن صورت  $T_{comm} = T_c$  خواهد بود. البته در این الگو برای جلوگیری از بن بست<sup>۵۲</sup>، پردازشها به دو گروه زوج و فرد تقسیم می‌شوند به‌طوری‌که ابتدا پردازشهای زوج و سپس پردازشهای فرد، کروموزومهای مهاجر را ارسال می‌کنند. بنابراین  $T_{comm} = 2T_c$  خواهد شد.

زمان اجرای این الگو پس از وقوع اولین مهاجرت برابر است با

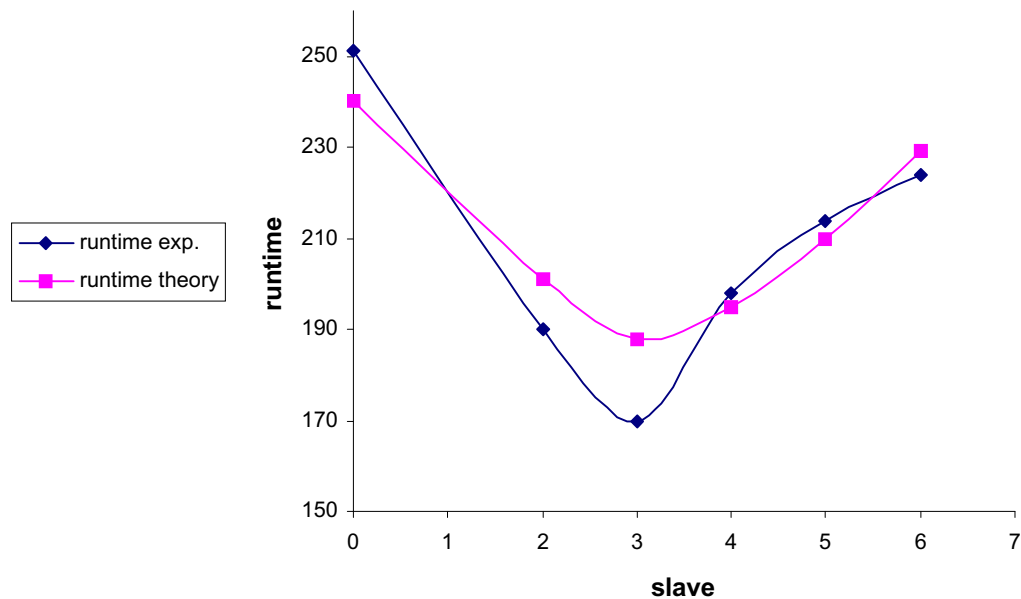
$$T_p = g_c \frac{n}{p} T_f + 2T_c \quad (۳)$$

که در آن  $g_c$  برابر است با تعداد نسلا بین دو مهاجرت، اگر مهاجرت از نوع همگام باشد و یا برابر است با تعداد نسلا تا زمانی که زیر جمعیت به همگرایی برسد، اگر مهاجرت از نوع غیرهمگام باشد.

زمان اجرای کلی این اسکلت برابر است با

$$T_p = k(g_c \frac{n}{p} T_f + 2T_c)$$

که در آن  $k$  برابر تعداد دفعاتی است که باید مهاجرت انجام شود تا کیفیت راه حل به دست آمده در این الگو با یک الگوریتم ژنتیکی سری هم ارز شود. در این الگو Speedup نیز برابر معادله زیر است:



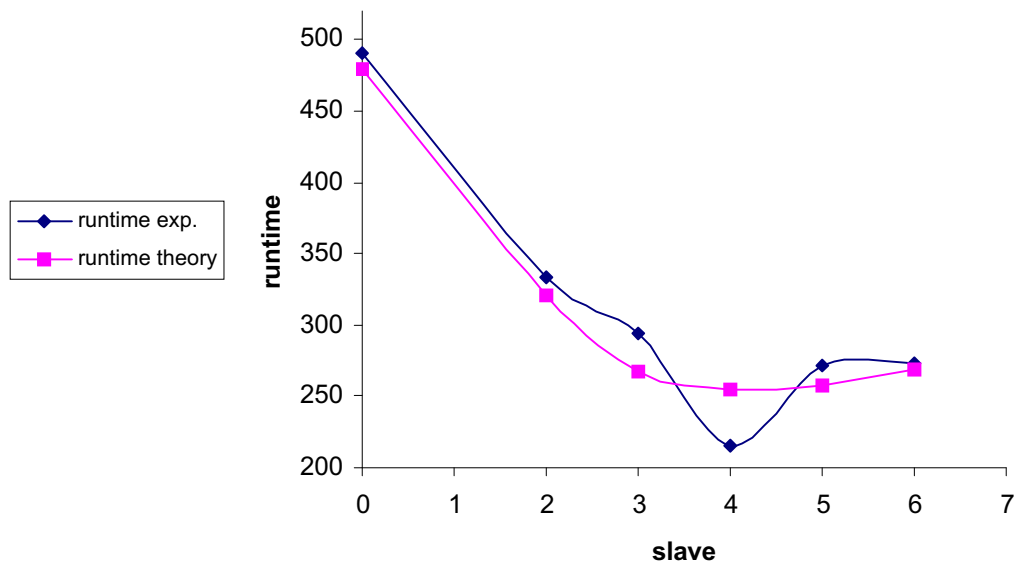
slave	runtime exp.	runtime theory
0	251	240
2	190	201
3	170	188
4	198	195
5	214	210
6	224	229

شکل ۷- زمان اجرای الگوی Master-SlaveGA به ازای  $T_f = 2ms$

محاسبه  $T_c$  در این الگو با توجه به آنکه  $T_c$  زمان ارسال پیامها بین ارباب و بنده است و چون زمان ارسال یک پیام از ارباب به کارگرهای متفاوت است. لذا زمان ارسال اطلاعات بین ارباب و هر یک از کارگرها در بنده جداگانه اندازه‌گیری شده و سپس متوسط آنها محاسبه شده است. آنگاه آزمایش ۱۰۰ بار انجام شده و متوسط بین این متوسطها به عنوان  $T_c$  انتخاب شده است. نتایج حاصل زمان واقعی اجرای الگوریتم در محیط انتقال پیام با استفاده از نرم‌افزار ام‌پی‌آی است و هیچگونه شبیه سازی انجام نشده است.

شکل (۹) نمودار زمان اجرای الگوی Multi-DemeGA را به ازای  $T_f = 2ms$  نشان می‌دهد. در این آزمایش الگوی

بهینه بنده‌ها برابر ۳ خواهد شد. به همین دلیل در نمودار زمان اجرای این الگو با افزایش تعداد بنده‌ها و ثابت نگه داشتن اندازه مسئله، به ازای ۳ بنده مسئله کمترین زمان اجرا را خواهد داشت. شکل (۸) نمودار زمان اجرای همین الگو را به صورت تجربی و تئوری به ازای  $T_f = 4ms$  نشان می‌دهد. در این آزمایش بنا به معادله (۲) تعداد بهینه بنده‌ها برابر ۴ خواهد شد و در این نقطه مسئله کمترین زمان اجرا را دارد. اما با افزایش تعداد بنده‌ها پس از این نقطه به دلیل افزایش سربار ارتباطی بین پردازشها زمان اجرا افزایش می‌یابد. با مقایسه دو شکل (۷) و (۸) به‌طور تجربی نیز ثابت می‌شود رفتار الگوریتم با افزایش زمان تابع ارزیابی بهتر می‌شود همان‌طور که در بحث مدل کارایی الگو نیز به‌طور تئوری به این نتیجه رسیدیم. برای



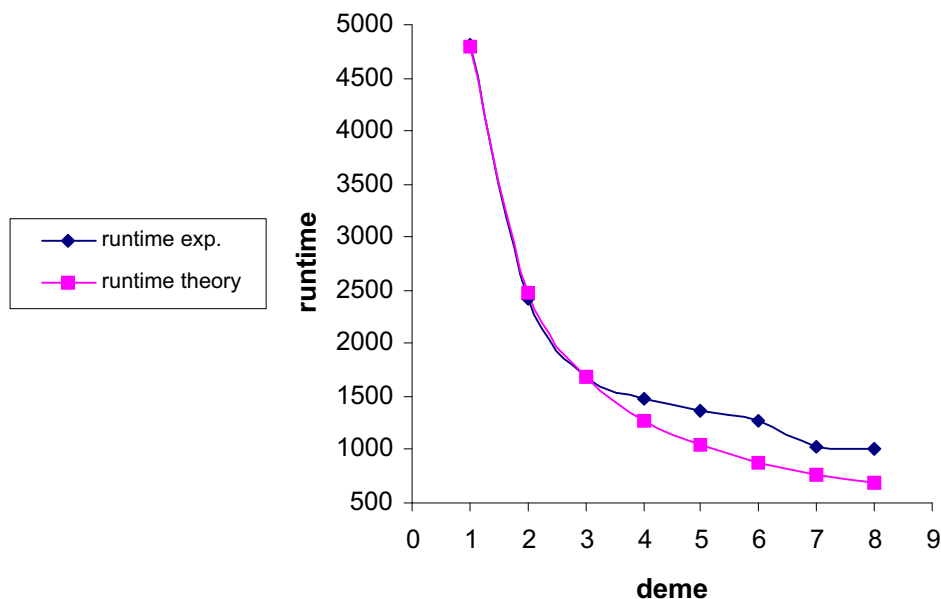
slave	runtime exp.	runtime theory
0	491	480
2	333	321
3	294	268
4	216	255
5	272	258
6	273	269

شکل ۸- زمان اجرای الگوی Master-SlaveGA به ازای  $T_f = 4ms$

و اثر آن را در کارایی پیدا کرد و بهترین الگوی مهاجرت را در زمان اجرا تعیین کرد که از مزایای این اسکلت است. در این آزمایش  $T_c$  زمان ارسال کروموزومها بین جزیره‌های مختلف است و مشابه مورد فوق اندازه‌گیری شده است و تقریباً برابر ۲۰ میلی ثانیه است. چون زمان ارسال پیام بین دو جزیره با زمان ارسال پیام بین ارباب و کارگرها در بنده متفاوت است لذا  $T_c$  های آنها نیز متفاوت است.

توجه شود هدف ما در آزمایشات، نشان دادن بهبود نسبی زمان اجرای مسئله به صورت موازی با چند پردازشگر، نسبت به اجرای آن به صورت سری با یک پردازشگر است و مقایسه الگوها با هم مد نظر نبوده است. بر روی هر نمودار محل تقاطع نمودار با محور عمودی، زمان اجرای الگوریتم را به صورت سری، به ازای یک پردازشگر نشان می‌دهد. ملاحظه می‌شود با

مهاجرت از نوع همگام است و مهاجرت هر ۱۰ نسل یکبار اتفاق می‌افتد. همچنین در هر بار مهاجرت ۱۰ کروموزوم از بهترین کروموزومهای هر زیر جمعیت، انتخاب می‌شوند. دلیل انتخاب الگوی مهاجرت همگام، همگامی با استفاده از همگامی سدی که یک شئوۀ آزاد از بن بست است انجام می‌گیرد. ولی در الگوی مهاجرت ناهمگام که یک الگوریتم موازی‌سازی کنترلی است الگوریتم باید به نحوی آزاد از بن بست طراحی شود. این نوع همگامی را ام-پی-آی انجام نمی‌دهد. بلکه باید در سطح طراحی الگوریتم انجام گیرد که در این صورت طراحی الگوریتم مشکل خواهد شد و کارایی الگوریتم نیز کاهش می‌یابد. همچنین در الگوی مهاجرت همگام برنامه نویس می‌تواند تعداد دفعات مهاجرت بین جزیره‌ها را تعیین کند با استفاده از این امکان می‌توان تعداد مهاجرت را تغییر داد



deme	runtime exp.	runtime theory
1	4810	4800
2	2420	2480
3	1680	1680
4	1470	1280
5	1360	1040
6	1270	880
7	1020	760
8	1010	680

شکل ۹- زمان اجرای الگوی Multi-DemeGA به ازای  $T_f = 2ms$  پس از وقوع دومین مهاجرت

## ۸- نتیجه گیری

با توجه به استفاده گسترده از الگوریتم های ژنتیکی در حل مسائل بهینه سازی و ضرورت موازی سازی آنها با توجه به دلایل یادشده، در این مقاله مجموعه ای از الگوها برای الگوریتم های ژنتیکی موازی معرفی شد. الگوهای معرفی شده در این مقاله کلی است و با مقداردهی مناسب پارامترهایشان برای انواع کاربردها قابل استفاده اند. با استفاده از این الگوها می توان برای بهینه سازی مسائل پیچیده و زمانبر، زمان اجرای برنامه را نسبت به استفاده از الگوریتم ژنتیکی سری برای حل همان مسئله به مقدار قابل توجهی کاهش داد. علاوه بر آن با

افزایش تعداد پردازشگرها، زمان اجرا تا تعداد پردازشگرهای مشخصی در الگوی Master-SlaveGA کاهش می یابد و سپس مجدداً افزایش می یابد. در شکل (۸) به ازای ۴ پردازشگر حد عملی از تئوری بهتر است که این امر نشان می دهد حجم سربار ارتباطی نسبت به حجم محاسبات بسیار کمتر بوده و با تقسیم بار محاسباتی بین ۴ پردازشگر که همان تعداد پردازشگر بهینه است نمودار زمان اجرا افت بیشتری داشته است.

در شکل (۹) به دلیل حجم بسیار کم ارتباطات بین پردازشهای موازی نسبت به حجم محاسباتی، زمان اجرا با افزایش تعداد پردازشگرها مرتباً کاهش می یابد.



تئوری بیشتر در زمینه این الگوها باز است. به عنوان مثال در الگوی Multi-DemeGA در زمینه انتخاب مقادیر بهینه برای تعداد مهاجرین، فرکانس مهاجرت، استراتژی انتخاب و جایگزینی مهاجرین هنوز تحقیقی انجام نشده است.

استفاده از مدل کارایی هر الگو، می توان زمان اجرای برنامه نوشته شده را قبل از اجرای آن تخمین زد. از دیگر ویژگیهای الگوهای معرفی شده، نقش برنامه نویس در زمان اجرای برنامه موازی با تعیین توپولوژی مجازی پردازشها و نیز ترکیب الگوهای پیشنهادی است. البته راه برای انجام دادن مطالعات

## واژه نامه

- |                         |   |                                   |
|-------------------------|---|-----------------------------------|
| 1. algorithmic skeleton | 20. Rabhi   | 34. pipeline                      |
| 2. performance model    | 21. evaluation  | 35. MPICH                         |
| 3. IBM                  | 22. fitness   | 36. 36.cluster                    |
| 4. BLUE GENE            | 23. selection   | 37. ethernet                      |
| 5. scable               | 24. crossover   | 38. simple genetic algori         |
| 6. MPI                  | 25. mutation  | 39. steadystate genetic algorithm |
| 7. PVM                  | 26. master-slave parallel genetic algorithm                                   | 40. one point crossover           |
| 8. data placement       | 27. multi-deme parallel genetic algorithm / island parallel genetic algorithm | 41. two point crossover           |
| 9. load balancing       | 28. migration   | 42. uniform crossover             |
| 10. synchronization     | 29. migration interval  | 43. roulette wheel selection      |
| 11. communication       | 30. hypercube   | 44. coordinator                   |
| 12. Murray Cole         | 31. fine-grain parallel genetic algorithm                                     | 45. worker                        |
| 13. host language       | 32. mesh  | 46. ring                          |
| 14. map                 | 33. hierarchical parallel genetic algorithm                                   | 47. deme                          |
| 15. farm                |   | 48. synchronous                   |
| 16. master-slave        |   | 49. asynchronous                  |
| 17. divide and conquer  |   | 50. template                      |
| 18. Darlington          |   | 51. 51.header                     |
| 19. Pelagatti           |   |                                   |

## مراجع

1. Sterling P., Becker B., Savarese D., et al, "BEOWULF: A Parallel Workstation for Scientific Computation," Proceeding of the 1995, International Conference on Parallel Processing (ICPP), Vol. 1, pp. 11-14, August 1995.
2. Monien B., Diekmann R., Feldmann R., Klasing R., luling R., Menzel K., RomkeT, and Schroeder U.-P., "Efficient Use of Parallel & Distributed Systems from Theory to practice," *Computer Science Today*, Springer LNCS 1000, pp. 62-77, 1995.
3. Cole, M.I., *Algorithmic Skeletons: Structured Management of Parallel Computation*, Pitman/MIT, London, 1989 .
4. Darlington J., Guo Y, To H.W., and Yang J., "Functional Skeletons for Parallel Coordination," proceeding of 1st EuroPar Conference, Stockholm, Sweden, pp. 55-66, Agust, 1995.
5. Gorlatch S., and Pelagatti, S., "A Transformational Framework for Skeleton Programs: Overview and Case Study," *Proceeding of HIP'99, a Satellite IPPS'99 Workshop*, San Juan, Puerto Rico, USA, LNCS 1586, pp. 123-137, April 1999.
6. Parsons, P.J., and Rabhi F.A., "Generating Parallel Programs from Skeleton Based Specifications," *Journal of Systems Architecture*, Vol. 45, pp. 261-283, 1998.
7. Holland J., "Genetic Algorithms and the Optimal Allocation of Trials, " *SIAM Journal on Computing*, Vol. 2(2), pp. 88-105, 1973.
8. Forrest S., "Genetic Algorithms," *Proceeding of the Fifth International Conference on Genetic Algorithms*, pp. 660, 1996.
9. Withley D, *Foundations of Genetic Algorithms 2*, pp. 45-62, Morgan Kaufmann, San Mateo, 1993.

10. Cantu-Paz E., "A Survey of Parallel Genetic Algorithm," *Calculateurs Paralleles, Reseaux et Systems Repartis*, Vol. 10(2), pp. 141-171, 1998.
11. Alba E., and Troya J.M., "A Survey of Parallel Distributed Genetic Algorithms", Internal T.R., University of Malaga, 1998.
12. Cantu-Paz E., "Designing Accurate and Efficient Parallel Genetic Algorithm," PhD Thesis, University of Illinois, 1999.
13. Adamidis P., "Review of Parallel Genetic Algorithms Bibliography," Internal T.R., Aristotle University of Thessaloniki, November 1994.
14. Gordon V.S., and Whitley D., "Serial and Parallel Genetic Algorithm as Function Optimizers," Proceeding of the 5<sup>th</sup> ICGA, pp. 177-183, 1993.
15. Lin S., Punch W.F., and Goodman E.D., "Coarse-Grain Parallel Genetic Algorithms: Categorization and New Approach," *Parallel & Distributed Processing*, October 1994.
16. Mejia-Olvera M., and Cantu-Paz E., "DGENESIS-Software for Execution of Distributed Genetic Algorithms," Proceeding of XX Conferencia Latinoamericana de Informatica, pp. 935-946, Monterrey, Mexico, 1994.
17. Gorges-Schleuter M., "ASPARAGOS An Asynchronous Parallel Genetic Optimization Strategy," Proceeding of the 3<sup>rd</sup> ICGA, Schaffer J.D.(ed.), Morgan Kaufmann, pp. 422-427, 1989.
18. Adamidis P., and Petridis V., "Co-operating Populations with Different Evolution Behavior," Proceeding of the second IEEE Conference on Evolutionary Computing, pp. 188-191, 1996.
19. Davidor Y., "A Naturally Occurring Niche & Species Phenomenon: The Model And First Results," Proceeding of the 4<sup>th</sup> ICGA, Belew R.K., Booker L.B. (eds.), pp. 257-263, 1991.
20. Robbins G., "EnGENEer- The Evolution of Solutions," Proceeding of the 5<sup>th</sup> Annual Seminar on Neural Networks and Genetic Algorithms, 1992.
21. Goodman E.D., "An Introduction to GALOPPS v3.2," TR#96-07-01, GARAGe, I.S.Lab, Dpt. Of C.S. and C.C.C.A.E.M., Michigan State University, East Lansing, 1996.
22. Potts J.C., Giddens T.D., and Yadav S.B., "The Development and Evaluation of an Improved Genetic Algorithm Based on Migration and Artificial Selection," *IEEE Transaction on Systems, Man, and Cybernetics*, 24(1), pp. 73-86, January 1994.
23. Herrera F., and Lozano M., "Gradual Distributed Real-Coded Genetic Algorithms," Technical Report # DECSAI-97-01-03, February, 1997. (Revisited Version 98).
24. Whitley, D., and Starkweather T., "GENITOR II: a Distributed Genetic Algorithm," *J. Expt. Theor. Artif. Intelligence* 2, pp. 189-214, 1990.
25. Voigt H.M., Santibanez-Koref I., and Born J., "Hierarchically Structured Distributed Genetic Algorithms," Proceeding of the International Conference Parallel Problem Solving from Nature, North-Holland, Amsterdam, pp. 155-164, 1992.
26. Ribeiro Filho, J.L., Alippi C., and Treleaven P., "Genetic Algorithm Programming Environments," *Parallel Genetic Algorithms: Theory & Applications*, Stender J. (eds), IOS Press, 1993.
27. Levine D., "Users Guide to PGAPack Parallel Genetic Algorithm Library," T.R. ANL-95/18, January 31, 1996.
28. Radcliffe N.J., and Surry P.D., "The Reproductive Plan Language RPL2: Motivation, Architecture and Applications," *Genetic Algorithm in optimization, Simulation and Modeling*, Stender J., Hillebrand E, Kingdon J. (eds.), IOS Press, 1994.
29. Erickson J.A., Smith R.E., and Goldberg D.E, SGA-Cube, "A Simple Genetic Algorithm for nCUBE 2 Hypercube Parallel Computers," TCGA Report No. 91005, the University of Alabama, 1991.
30. Michaelson G., Scaife N., Bristow P., and King P., "Nested Algorithmic Skeletons from Higher order Functions," Submitted for parallel Algorithms and Applications, Special Issue on High Level Models and Languages for parallel processing, September 2000.
31. Scaife N., Bristow P., and Michaelson G., "Engineering a Parallel Compiler for Standard ML," in Proceeding of the 10<sup>th</sup> International Workshop on Implementation of Functional Languages, IFL '98, University College London, pp. 213-225, September 1998.
32. Skellicorn D.B., and Pelagatti S., "Building Programs in the Network of Task Model," Proceeding of the ACM Symposium on Applied Computing (SAC2000), Como, Italy, pp. 248-254, March 2000.
33. Hachler G., "ALWAN: Design and Implementation of a parallel Coordination Language," PhD thesis, Department of Computer University, Basel University, 1997.

34. Herrmann C.A., Lengauer C., Gnz R.,Laitenberger J., and Schaller C.,“ A Compiler for HDC,” Technical Report MIP-9907,Fakultt fr Mathematik und Informatik , Universitt Passau, May 1999.
35. <http://lancet.mit.edu/ga>
36. <ftp://ftp.mcs.anl.gov/pub/mpi/nt/mpich.nt>
37. Munetomo, Taakai Y., and Sato Y.,“An efficient migration scheme for SubPopulation-Based Asynchronously Parallel Genetic Algorithms,” Internal T.R., 1993.
38. Whitely D., Rana S., and Heckendorn R.B.,“Exploiting Separability in Search: the Island Model Genetic Algorithm,”*Journal of Computing and Information Technology* , 1999.