علوم و مهندسی کامپیوتر
نشریه علمی پژوهشی انجمن کامپیوتر ایران
مجلد ۷، شماره ۱ و ۳ (الف)، ۱۳۸۸
صفحات ۴۱-۴۸
مقاله عادی

ارائه یک الگوریتم ژنتیک برای زمانبندی شبکه‌های گرید با معماری سلسه مراتبی

شیرین کریمی محمدکاظم اکبری سیدمحمدرضا میرزابابائی

دانشکده مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه صنعتی امیرکبیر، تهران، ایران

چکیده

یکی از اهداف اساسی در شبکه‌های گرید محاسباتی بالاتر بردن قدرت پردازش است. با توجه به گستردگی فراوان منابع بدون مشخص بودن سازمان اصلی درخواست دهنده کار، استفاده از تکنیک‌هایی به منظور کاهش زمان اجرای یک برنامه کاربردی و میزان استفاده بهینه از منابع و در نتیجه کارایی هر چه بیشتر می‌تواند ما را به هدف نزدیکتر سازد. به این منظور وجود سیستم‌های زمانبندی قدرتمند در این شبکه‌ها بسیار ضروری است. در این مقاله ابتدا به ارائه یک مدل معماری برای زمانبندی پرداخته‌ایم و سپس با هدف کاهش زمان اجرا و انتخاب بهترین مجموعه از این Max-Min و قیاس با دو روش لیست و NPB منابع از ترکیب دو الگوریتم ژنتیک و لیست استفاده کرده‌ایم که در نهایت با تست سیستم توسط انواع مختلف برنامه‌های کاربردی منطبق بر نتیجه حاصل گردید که روش فوق بهبود زمانی قابل قبولی را ارائه کرد، اما در برابر زمان اجرای الگوریتم افزایش یافت. لذا با در نظر گرفتن هر دو پارامتر چنانچه کارهای با حجم متوسط و بالا توسط این روش زمانبندی گردند، نتیجه بسیار مطلوبی حاصل می‌گردد.

کلمات کلیدی: گرید محاسباتی، الگوریتم‌های ژنتیک، زمانبندی لیست، معماری سلسله مراتبی.

۱- مقدمه

گرید سیستمی مبتنی بر استفاده اشتراکی از منابع است که در آن منابع متعلق به چندین سازمان مختلف هستند.

گرید یک سیستم ناهمگن[1] توزیع شده [۱] است و بنابراین هیچ کنترل مرکزی مطلقی بر روی آن وجود ندارد [۲]. در این محیط یک یا چند کاربر می توانند از منابع به صورت اشتراکی استفاده کنند بدون اینکه لازم باشد بدانند منابع متعلق به چه سازمانی است. گرید عموماً یک ساختار محاسباتی دارد بنابراین در هر مرحله باید تخصیص منابع به بهترین نحو ممکن انجام شود، طوری که حداکثر استفاده از منابع صورت بگیرد. به این منظور وجود یک سیستم میانی[2] که بعنوان یک سیستم نرم افزاری زیر ساخت گرید تلقی می گردد در توسعه پروتکل‌ها و عملکرد واسط[3] های استاندارد موثر می باشد. با توجه به مطالب فوق چنانچه هدف بالاتر رفتن میزان کارایی و کاربرد بهینه از منابع[4] در سیستمهای گرید باشد، زمانبندی کار[5] های وارده به گرید یک نقش کلیدی بازی خواهد کرد [۳]. لذا در زمانبندی نکته‌ای که باید مد نظر قرار گیرد میزان همخوانی کارها و منابع محاسباتی در دسترس و استفاده بهینه از منابع ، در حین اجرای یک برنامه کاربردی می باشد.

۲ - ارائه معماری اولیه

در زمانبندی یکی از نکاتی که شدیداً مورد توجه قرار می گیرد این است که سیستمی که برنامه کاربردی بر روی آن اجرا می‌شود تا حد امکان یک محیط هوشمند باشد. مبحثی که در گرید برای ما بسیار حائز اهمیت است مدیریت منابع در این محیط است [۴] به این معنی که باید در طراحی معماری به این نکته نیز توجه داشته باشیم که مدیریت منابع نیز تا حد امکان هوشمند باشد و در ضمن بتواند توابعی را جهت کشف منابع، انتشار منابع، زمانبندی، پذیرش و روئیت کارها داشته باشد. علاوه بر این توابع مورد نظر باید توابعی باشند که در سطح میانی استاندارد شده باشند [۵].

بر اساس بررسی ها و مطالعات انجام شده سرویس زمانبندی در صورتیکه نگرش بالا به پایین داشته باشد قابلیت مدیریت بالاتری در سطح میانی خواهد

داشت به این ترتیب سرویس ارائه شده قادر خواهد بود که نیازمندی‌ها را از سطح بالا بنگرد. از طرف دیگر باید با ایجاد یک رابط میانی مناسب، قدرت محاسباتی گرید را بالا برد که این افزایش در واقع منجر به بالاتر رفتن کارایی در کل سیستم می گردد و از آنجا که گرید به علت خاصیت ناهمگونی منابع پیچیدگی‌های خاص خود را دارد طبیعی است که پیچیدگی سیستم زمانبندی تحت تأثیر قرار می‌گیرد. علاوه بر این مورد پیچیدگی برنامه کاربردی و نیازمندی‌های کاربر را نیز داریم که باید لحاظ گردند. تمام موارد فوق و مصادیق آنها به طور مستقیم کارایی را تحت تأثیر قرار می‌دهند بنابراین به راحتی می‌توان به این نتیجه رسید که زمانبندی استفاده شده در این میان نیز بر روی کارایی اثر مستقیم دارد.

زمانبندی برنامه‌های کاربردی فقط شامل انتخاب مناسب منبع برای کار مورد نظر نمی‌باشد بلکه نکات دیگری را نیز باید در نظر گرفت از جمله QoS مورد نظر و درخواست شده توسط کاربر با لحاظ کردن تمامی پارامترهای دیگر QoS (QoS دستگاهها، QoS شبکه و..). علاوه بر این نکته بسیار مهم دیگری که یک زمانبند باید به آن توجه داشته باشد ارتباط بین کارهای[۶] مختلف در مقاطع زمانی[۷] متفاوت است. در ضمن تغییرات دینامیک در شبکه نیز باید لحاظ گردد. پس به طور خلاصه می توان چنین گفت که یک زمانبند باید قادر باشد با ایجاد توازن بر روی یک سری توابع بهینه سازی تمام فعالیت‌های فوق را تا حد معقول انجام دهد به عنوان مثال به تقاضاهای کاربر نظیر هزینه، زمان پاسخ و یا محدودیت‌های مختلف منابع نظیر میزان استفاده از منابع[۸] و یا بازدهی منابع[۹] پاسخ دهد.

با توجه به مواردی که مطرح گردید و به منظور جلوگیری از تداخل منابع[۱۰]، طبق بررسی های انجام شده چنانچه تقسیم بندی بین انواع کارهای ورودی به سیستم گرید انجام شود و زمانبند بر اساس نوع کار [۶] تصمیم به زمانبندی بگیرد نتایج مناسب‌تری به دست می آید.

ایده اولیه برای این طرح مبتنی بر دو روش است:

- کشف منابع توزیع شده و تخصیص منابع به صورت محلی.
- استفاده از یک انبار مرکزی بر روی منابع.

راه حل اول روشی است که در مورد کارهای کوچکی که به راحتی توسط گرید پذیرفته می‌شوند مناسب است، اما راه حل دوم بیشتر مناسب کارهای دوره‌ای است که در نهایت بهینه‌سازی‌هایی را به دنبال خواهد داشت. در ایده اولیه از راه حل اول تحت عنوان زمانبند محلی[۱۱] نام می‌بریم و راه حل دوم را ابرزمانبند[۱۲] یا زمانبند سرتاسری[۱۳][۷] می نامیم. لازم به ذکر است که زمانبند سرتاسری از نظر موقعیت منطقی در سطح بالاتری نسبت به زمانبند محلی قرار دارد.

در ادامه تقسیم‌بندی کارها بر روی گرید لازم است به این نکته اشاره شود که گرید عموماً بر روی زمان اجرا حساسیت خاصی دارد که در واقع یکی از دلایل آن نیز دینامیک بودن محیط است که در هر لحظه زمانبندی و زمانبندی مجدد را تحت تأثیر قرار می‌دهد.

با این توضیح می دانیم که کارها با حجم[۱۴] و نوع[۱۵] مختلف وارد محیط محاسباتی می گردند. حال چنانچه بخواهیم تفسیم بندی را فراتر از کارهای کوچک و بزرگ داشته باشیم می‌توانیم از دسته بندی تحت عناوین کارهای منفرد، کارهای متوسط با وابستگی به هم و کارهای همکار، استفاده نماییم.

از طرف دیگر علاوه بر اینکه لازم است برای هر دسته از کارهای فوق از سطوح مختلف زمانبندی استفاده کنیم، سطوح پیاده سازی الگوریتم‌ها نیز باید مد نظر قرار گیرد. طرح ارائه شده نهایی از دو سطح به این منظور استفاده می‌کند [۸] که توضیحات مبسوط در خصوص این دو سطح در بخش‌های بعدی خواهد آمد. این دو سطح شامل:

- زمانبندی در سطح سیستم: زمانبندی کارهای منفرد و بدون وابستگی را به عهده دارد.
- زمانبندی در سطح برنامه کاربردی: زمانبندی کارهای وابسته و اشتراکی را به عهده دارد.

هدف اساسی از این تقسیم بندی این است که کارهای با حجم کم و منابع محدود چنانچه بخواهند وارد یک الگوریتم زمانبندی مرکزی هر چند دقیق شوند نمی‌توانند نتیجه مناسبی داشته باشند چرا که زمانی که صرف اجرای الگوریتم زمانبندی می‌گردد بیشتر از زمان اجرای یک کار کوچک خواهد بود. بنابراین با توجه به توضیحات قبلی بهتر است کارهای کوچک و مستقل در سطح سیستمی و توسط یک زمانبند محلی زمانبندی گردند. اما چنانچه وابستگی بین کارها وجود داشته باشد [۹] و همینطور برنامه کاربردی حجم بالایی داشته باشد استفاده از زمانبند سرتاسری نتیجه بهتری خواهد داشت.

مراحل کار به این شکل است که ابتدا به یک تعریف اولیه می‌پردازیم، سپس به طرح اولیه اشاره‌ای خواهد شد، در ادامه معماری‌های مختلف زمانبندی گرید عنوان می‌گردد و در نهایت معماری نهایی و طرح اصلی معرفی می شود. در جهت اهداف فوق طرح اولیه‌ای ارائه شده است، بدین ترتیب که تعدادی کار داریم و تعدادی نود، که این کارها باید بر روی نودها اجرا شوند.

هر نود خود شامل مجموعه‌ای از منابع ناهمگن است. کارها بر روی استفاده از منابع سخت افزاری و زمان اجرا محدودیت دارند به این معنی که یک کار باید براساس درخواست کاربر بر روی منبع خاص و با شرایط خاص اجرا شود. نودهای گرید به این شکل عمل می کنند که منابع محاسباتی محلی را خودشان مدیریت می کنند و به این منظور نیز از سیاست‌های محلی استفاده می‌نمایند. لذا استفاده از یک الگوریتم که روش پیاده‌سازی راحتی دارد می‌تواند در این سطح به نیاز کاربر پاسخ بدهد. در این طرح از الگوریتم لیست برای زمانبندی در این سطح استفاده شده است.

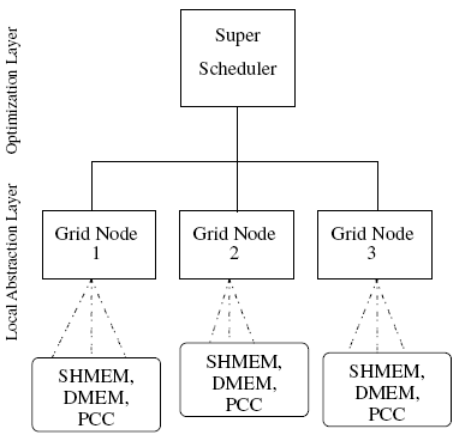
نودهای گرید درحقیقت مجموعه‌ای از منابع با دسته‌بندی‌های مختلف می‌باشند، این منابع بر اساس انواع معماری‌های مختلف، به عنوان مثال حافظه مشترک[۱۶] حافظه توزیع شده[۱۷] و کلاسترهای محلی دسته‌بندی شده‌اند.

حال با فرض اینکه تمامی کارها نتوانند بر روی منابع محلی خودشان اجرا شوند در این صورت زمانبندی و تخصیص منابع به بخشی تحت عنوان زمانبند مرکزی یا سرتاسری[۱۸] واگذار می‌شود. هدف اصلی آن نیز یافتن مناسب ترین ترتیب از تخصیص کارها به منابع است.

مطالبی که عنوان شد در واقع اشاره‌ای بود به معرفی معماری دو سطحی برای زمانبندی در برابر مدل‌های مختلف دیگری که وجود دارد [۱۰]، مانند استراتژی زمانبندی مرکزی[۱۹] و یا توزیع شده[۲۰].

در زمانبندی دو سطحی در سطح اول زمانبند محلی را داریم و در سطح دوم زمانبند مرکزی. زمانبند محلی کار را می‌گیرد و بر اساس سیاست‌ها و دیگر اطلاعات محلی منبع مورد درخواست را به کار مورد نظر اختصاص می‌دهد.

در شکل زیر نمای کلی این طرح ارائه شده است.



شکل ۱- معماری زمانبند مرکزی

در پیاده‌سازی زمانبند مرکزی می توان از شیبه سازهای مختلف استفاده نمود. اما در تمام موارد یک سری محدودیت‌ها وجود دارد که مستقل از شبیه‌ساز هستند. با در نظر گرفتن و لحاظ کردن این محدودیت‌ها جواب بهینه قابل دسترس‌تر می‌باشد. این محدودیت‌ها را می‌توان در حالت کلی به شکل زیر دسته‌بندی نمود:

- خواص مربوط به کار ورودی به سیستم: پارامتری است که بستگی به برنامه کاربردی دارد که قرار است بر روی شبکه اجرا شود.
- خواص مربوط به محیط گرید: ساختار منطقی و فیزیکی شبکه را شامل می‌شود و مبتنی بر قدرت محاسباتی است که اجزاء و ماشین‌های موجود در شبکه می‌توانند آنرا پشتیبانی کنند.
- خصوصیات داده ها و اطلاعات توزیع شده بر روی گرید: شامل انواع مختلفی از داده‌ها و اطلاعاتی است که در مورد سیستم موجود می‌باشد.

۲-۱- طرح اصلی

با توجه به موارد فوق، یکی از اهداف ما این است که کاربر محدودیتی جهت درخواست کار نداشته باشد. به عبارتی دیگر کاربر باید بتواند در سطوح مختلف تقاضای اجرای کار را داشته باشد [۱۱ و ۱۲].

نیاز دیگری که در مراحل مختلف به آن اشاره شد استفاده از زمانبند مرکزی

است که بتواند یک مدیریت از بالا به پایین داشته باشد.

به دلایل فوق نیاز ما به یک معماری با ساختار کنترل مرکزی و به صورت سلسله مراتبی می باشد که کاربر نیز محدودیت خاصی جهت تقاضای کار نداشته باشد. در نتیجه روش مورد استفاده نهایی [۱۳] مبتنی بر روش گرید محاسباتی با کارایی بالا[۲۱] است.

با این توضیحات در ادامه به بعضی از خصوصیات و شرایط کارهای موجود در یک شبکه و نیازمندی‌های آن می‌پردازیم.

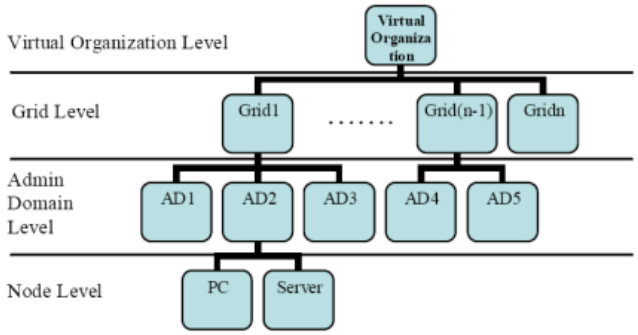
۱- دینامیک بودن محیط گرید [۱۴]: قدرت محاسباتی و سرویس‌های ارائه شده در گرید توزیع شده و دینامیک هستند . از طرفی منابع در دسترس روی هر نود گرید در لحظات مختلف متفاوت است که این فاکتور قدرت گرید را در لحظات مختلف تحت تأثیر قرار میدهد. به عنوان مثال وقتی یک سرویس جدید به سیستم و یک نود خاص افزوده می‌شود دیگر بخش‌ها نیز باید از این قابلیت جدید آگاه شوند. یک معماری سلسله مراتبی باید بتواند در هر سطحی از معماری این نیازمندی را برآورده سازد و در سطوح مختلف بر روی آن اعمال مدیریت مناسب بنماید.

۲- متفاوت بودن نودهای گرید و تعداد آنها: هر نود در سیستم سلسله مراتبی مستقل است. هر نود خودش می تواند تصمیم بگیرد که یک کار را اجرا بکند یا نه و یا اینکه از چه روشی برای زمانبندی استفاده کند تا میزان استفاده از منابع به مقدار بهینه خودش برسد.

۳- متفاوت بودن کارها و نیازمندی‌های آنها: کارهای مختلف در یک سیستم نیازهای متفاوتی دارند، به عنوان مثال اینکه یک کاربر روی چه منبعی با چه توان محاسباتی اجرا شود در برنامه های مختلف با هم فرق می کند. بعضی از کارها نیاز دارند به اینکه در حداقل زمان ممکن اجرا شوند، در برابر ممکن است نیازمندی اساسی کار دیگر حداقل هزینه باشد. با این توضیح از آنجا که گستردگی این تفاوت در یک سیستم توزیع شده بیشتر می شود بنابراین طبیعی است که باید یک دسته بندی بین کارها، انواع آنها و نیازمندی‌هایشان انجام شود.

۴- پذیرش گرید: تابع اصلی در گرید در حد یک سرویس میانی است. به منظور استفاده هر چه بیشتر از منابع ضمن برآوردن کیفیت سرویس تقاضا شده لازم است که سیستم خودش را با اطلاعاتی که از قبل موجود بوده تطبیق بدهد.

با این توضیحات و لحاظ کردن موارد فوق طرح پیشنهادی در شکل زیر نمایش داده شده است.



شکل ۲- معماری لایه‌ای پیشنهادی در گرید

نکته ای که در ابتدا لازم است به آن اشاره شود این است که در این معماری در سه سطح کاربر می تواند تقاضای اجرای کار را داشته باشد یعنی در سطح حوزه مدیریت[۲۲]، در سطح گرید و در سطح سازمان مجازی[۲۳].

سطح AD (حوزه مدیریت) در برگیرنده گروه‌هایی از ماشین‌هاست که تمام نودهای آن متعلق به یک سازمان هستند. به عنوان مثال در شکل۲ AD1 متعلق به مرکز کامپیوتر است و AD2 متعلق به سازمان علوم کامپیوتر است. پس AD از طرفی به عنوان یک سیستم کلی محسوب می‌شود و از طرفی فقط بر روی منابع خودش کنترل دارد و بر روی ADهای دیگر کنترلی ندارد.

یک گرید می تواند ADهای گسترده‌ای داشته باشد که باهم در ارتباط کاری خوب و مناسبی باشند. علاوه بر این هر گرید می‌تواند به صورت کاملاً مستقل از دیگر گریدها عمل کند برای مثال در شکل۲ گرید ۱ می‌تواند در ایران، گرید ۲ در چین و ... قرار داشته باشند.

بالاترین سطح نیز در برگیرنده تمامی نودهاست که مدیریت اصلی بر روی منابع در این سطح قرار دارد. حال هدف ما استفاده از الگوریتم ژنتیک به عنوان الگوریتم زمانبند مرکزی برای مدیریت کلیه منابع در این سطح می‌باشد. از الگوریتم زمانبندی لیست نیز به عنوان زمانبند محلی استفاده می‌کنیم. علت اصلی استفاده از الگوریتم لیست سادگی این الگوریتم و لحاظ شدن اولویت کارها در آن است. بعبارتی این الگوریتم در تعریف خود کاری را که اولویت بالاتری دارد ابتدا زمانبندی می‌نماید.

۳- معرفی الگوریتم کلی

همانطور که پیشتر اشاره شد یکی از اهداف ما از انتخاب این معماری این بود که کاربر از سطوح مختلف بتواند تقاضای اجرای کار داشته باشد. بنابراین با این توضیح کاربر از سه سطح می تواند تقاضای اجرای برنامه کاربردی داشته باشد که به شرح زیر می باشد:

۱. در سطح حوزه مدیریت تقاضای خود را مطرح کند.
۲. در سطح گرید تقاضای خود را مطرح کند.
۳. درسطح سازمان مجازی تقاضای خود را مطرح کند.

علاوه بر این تقسیم بندی دیگری نیز بر روی نوع کارها خواهیم داشت که شامل:

۱. کار به صورت منفرد و با حجم کم، بدون وابستگی به دیگر کارها و یا با حجم متوسط بدون وابستگی است.
۲. کار به صورت یک مجموعه وابسته است.

الگوریتم با در نظر گرفتن سطح درخواست کار توسط کاربر و نوع کار تصمیم به انتخاب نوع زمانبند می‌نماید. نکته‌ای که باید به آن توجه شود این است که الگوریتم ژنتیک با توجه به ساختاری که دارد الگوریتمی با زمان اجرای بالا و دقت بسیار بالاست. یعنی قادر است مناسب‌ترین مجموعه منابع در دسترس را فراهم آورد اما از نظر زمانی مقرون به صرفه نیست که کاری با حجم بایین را جهت زمانبندی به این الگوریتم واگذار نماییم. لذا روش استفاده شده در این تحقیق به این شکل است که کارهای با حجم کم و کارهایی که توسط کاربر در دو سطح پایینی تقاضا شده باشند را ابتدا توسط زمانبند محلی زمانبندی می‌نماید، در توضیح این مسأله باید گفت که نیاز کارهای وارد شده به سیستم مقدار مشخصی است، از طرفی امکان آگاهی از بار کاری منابع و نودهای گرید در هر لحظه از زمان مشخص است، زمانبند با مقایسه بین این دو مقدار می‌تواند به این نکته پی ببرد که آیا می‌توان کار ها را محلی زمانبندی کرد یا خیر؟ و در صورت منفی بودن جواب، زمانبند مرکزی با الگوریتم ژنتیک را فراخوانی می‌کند. اما کارهای با حجم بالا و در سطوح بالا از همان ابتدا توسط الگوریتم ژنتیک زمانبندی می‌شوند.

۴- الگوریتم ژنتیک مورد استفاده

تا کنون مطالعات زیادی بر روی الگوریتم های مختلف زمانبندی صورت گرفته است [۲، ۷ و ۱۵]. یکی از اهداف اساسی این الگوریتم ها یافتن مناسبترین مجموعه از منابع جهت زمانبندی می باشد. به عبارتی دیگر دستیابی به یک پاسخ بهینه و یا نیمه بهینه یکی از اهداف اساسی می باشد. در میان روشهای مختلف الگوریتم های ژنتیک در یافتن جواب بهینه بیش از موارد دیگر موفق بوده‌اند. این الگوریتم ها با الهام از طبیعت و اصل نگهداری و بقای نسل اصلح بهترین انتخاب را در اختیار ما قرار می دهد. دلایلی وجود دارد که با استناد به آنها می توان به قابلیت این دسته از الگوریتم ها بیشتر پی برد. این موارد شامل:

۱. الگوریتم ژنتیک با یک مجموعه کد شده از پارامترها عمل می کند نه با خود پارامترها.

۲. الگوریتم ژنتیک جستجو را بر روی یک مجموعه انجام می دهد نه یک نقطه منفرد.

۳. با استفاده از یک تابع مشخص در هر مرحله ارزیابی لازم را انجام می دهد.

۴. از قوانین احتمال استفاده می کند.

با این توضیحات طبیعی است که این خانواده از الگوریتم ها از دقت بالایی برخوردار باشند. این الگوریتم ها در حالت کلی به این صورت عمل می نمایند که ابتدا جستجو را بر روی یک فضا[۲۴] انجام می دهند. این فضا همان جمعیت اولیه است. الگوریتم درهر مرحله با ارزیابی مرحله قبل به کار خود ادامه می‌دهد. ارزیابی توسط تابع برازش صورت می‌گیرد. این تابع می‌تواند ملاک‌های متفاوتی داشته باشد به عنوان مثال زمان اجرا، کیفیت سرویس تقاضا شده توسط کاربر، میزان استفاده از منابع و ... باشد. در این تحقیق زمان اجرا به عنوان تابع ارزیابی مورد استفاده قرار می‌گیرد.

الگوریتم ژنتیک سه فاز اصلی دارد شامل:

۱. مرحله آغازین

۲. عبور

۳. جهش

نکته دیگری که باید لحاظ شود این است که شرطی را جهت اتمام الگوریتم باید در نظر گرفت. این شرط می تواند شامل رسیدن به جواب دلخواه، به اتمام رسیدن تعداد تولیدها و یک عامل خارجی باشد.

۴-۱- مرحله آغازین

اولین کاری که باید صورت بگیرد ایجاد یک جمعیت اولیه است. این جمعیت در واقع شامل تعدادی رشته است که با روش خاصی منابع را در اختیار گرفته‌اند. در واقع نمایانگر منبعی y نمایانگر کار مورد نظر و بعد x دارند که بعد y و x رشته‌ها دو بعد است که اجرای کار را به عهده گرفته است.هر کدام از این رشته‌ها متناظر با کروموزوم در دنیای واقعی است. تعداد رشته‌ها نیز اعضای جمعیت را تشکیل می دهند.

برای مرحله آغازین الگوریتم‌های مختلفی وجود دارد از جمله الگوریتم بر مبنای ارتفاع یا بر مبنای اولویت کارها و در نهایت الگوریتم‌های اکتشافی.

در این طرح از الگوریتم بر مبنای ارتفاع استفاده شده است که اساس آن مبتنی بر تعداد کارهایی است که باید قبل از یک کار خاص اجرا شوند. دو روش برای محاسبه ارتفاع وجود دارد.

۱- اگر یک کار قبل از خودش[۲۵] چیزی نداشته باشد ارتفاعش صفر است.

۲- در غیر اینصورت ارتفاع برابر با حداکثر ارتفاع کارهای ماقبل بعلاوه یک است.

الگوریتم در زیر آمده است.

```
Get_height(no input)
Start:
For (i=0,to no of task) do
{
If(current task has predecessor
and task's height has'nt been
found,find_height(i));
else if the height is 0;
}
End:
Find_height(input * task)
Start:
If(no predecessor) height=0;
Return(0);
Else If (has predecessor and
height if found)
Return(task's height);
Else
For(i=0; to no of task) do
Height= currentheight+(find_height(i)+1)
Current task's height=height
Return (task's height)
End.
```

شکل ۳- الگوریتم بر مبنای ارتفاع

۴-۲- عبور

در این مرحله رشته ها از دو بعدی به یک بعدی تبدیل می‌شوند. لذا یک نگاشت یک به یک برای اینکار صورت می‌گیرد و زیر رشته‌ها را در آرایه دو بعدی از انتها به هم وصل کرده و یک رشته طولانی بوجود می‌آورد. هدف اصلی از این مرحله این است که با تعویض مکان کارهایی که در مرحله آغازین صورت گرفته است ترتیبی را بدست آورد که هدف نهایی را بهینه سازد. برای اینکار الگوریتم‌های زیادی وجود دارد که در اینجا از الگوریتمی با عنوان عبور سیکلی استفاده شده است. این الگوریتم در ادامه آمده است.

۴-۲-۱- جابجایی با روش سیکل

این متد از دو روش برای جستجو و مقایسه استفاده می‌کند. در روش اول نقطه شروع را صفر قرار می‌دهد و در روش دوم از یک عدد تصادفی شروع به جستجو می‌نماید. بر اساس یکی از دو مورد کاری که در مکان مشخص شده در رشته اول وجود دارد را انتخاب و علامتدار می‌کند. سپس در رشته دوم به جستجوی همین کار پرداخته، آن را نیز علامتدار می‌کند. حال کار موجود در رشته اول در مکان متناظر به عنوان اولین کار انتخاب و علامتدار می‌شود. الگوریتم تا جایی ادامه می‌یابد که دوباره به اولین کار انتخاب شده برسد. پس از این مرحله رشته‌هایی که علامتدار شده‌اند در جای خود باقی می‌مانند و در غیر اینصورت اعضای دو رشته با هم جابجا می‌گردند. در پایان دو رشته دو بعدی شده و تابع برازش برای هر دو محاسبه می‌گردد. لازم به ذکر است که این عملیات باید برای تمام جمعیت صورت بگیرد. در زیر روش کلی این الگوریتم آمده است.

```
Cycle-Crossover(type,schedule1,schedule2)
Start
  Store_String(sched1);
  Store_String(sched2);
If (of type 1)
  Point=0;
Else if (of type 2)
  Point=random(pco1.getsize());
Get first task;
Do loop;
  Mark task as done;
  Get task in string 1 using current
  position in string2;
  If (current task=first task)
    Exit loop;
  End do loop;

For (i=0,no of task) do:
  If (task not in cycle)
  Swap task between two string;
  Reorder(1,sched1);
  Reorder(2,sched2);

  Restore(1,sched1);
  Restore(2,sched2);
End:
```

شکل ۴- الگوریتم عبور

۴-۳- جهش[۲۶]

آخرین مرحله جهش نهایی هر رشته است. هدف اصلی این است که با انجام این مرحله احتمال عدم تغییر در رشته‌های اولیه را به صفر برسانیم. برای اینکار نیز الگوریتم‌های متفاوتی وجود دارد که الگوریتم انتخابی در اینجا جهش افزایشی می‌باشد. در ادامه شرحی بر این الگوریتم داده می‌شود.

۴-۳-۱- جهش افزایشی[۲۷]

این الگوریتم یک رشته دو بعدی را به عنوان ورودی می‌گیرد. فرم هر رشته نیز ثابت می‌ماند بعنوان مثال اگر پردازنده ۳ در رشته اول، ۵ کار را در اختیار داشته باشد پس از این عملیات نیز کماکان ۵ کار را در اختیار خواهد داشت با این احتمال که ممکن است کارها همان کارهای اولیه نباشند. اساس کلی کار مبتنی بر انتخاب یک پردازنده به طور تصادفی و انتخاب یکی از کارهای آن پردازنده است. مکان پردازنده بعنوان مکان شروع جستجو جهت یافتن مناسب ترین مکان برای اضافه کردن این کار و حذف از مکان قبلی انتخاب شده و عملیات جستجو ادامه می‌یابد. این الگوریتم در زیر آمده است.

```
Add-mut(input schedule)
Start
Proc=random(no of processor)
Task=random(size of substring)
/*find new processor*/
Proc2=random(no of processor)
Find insert point;
/*insert task*/
Add_mut_insert(rand_no,sched_no,proc2,task);
/*remove old task*/
Add_mut_remove(location,sched_no,proc);
```

شکل ۵- الگوریتم جهش

پس از انجام اینکار بر روی تک تک رشته‌ها یکبار دیگر برازش انجام می‌شود. در صورتی که الگوریتم پایان یافته باشد مناسب‌ترین پاسخ به عنوان راه حل نهایی انتخاب می‌شود و گرنه مراحل ذکر شده ادامه می‌یابند.

۵- ارزیابی مدل و الگوریتم‌ها و نتایج عملی

به منظور بررسی معماری و شبیه‌سازی الگوریتم‌ها از ابزار Gridsimtoolkit-v4 استفاده شد. با استفاده از این محیط ۴ نود گرید ساخته شده است که در مجموع شامل ۳۶ پردازنده می‌باشند که با ساختار سلسله مراتبی با هم ارتباط محکم دارند و تأخیرهای معمول شبکه در آن لحاظ شده است [۱۶]. از الگوریتم ژنتیک به عنوان الگوریتم مرکزی و از الگوریتم لیست به عنوان الگوریتم محلی استفاده شده است [۱۷ و ۱۸]. در مورد الگوریتم ژنتیک نیز مقادیر زیر لحاظ گردید:

- جمعیت اولیه: ۲۰
- مرحله آغازین: استفاده از الگوریتم ارتفاع
- مدل عبور: عبور سیکلی با احتمال ۶,۰
- مدل جهش: جهش افزایشی با احتمال ۱,۰
- تعداد تولید نسل: ۲۰

قبل از اینکه به بررسی نتایج حاصل از اجرای الگوریتم پرداخته شود لازم است توضیحاتی در خصوص کارهایی که جهت اجرا به این سیستم واگذار شده‌اند، داده شود. در تست‌های انجام شده در حالت کلی از ۷ گروه برنامه استفاده شده که از حجم کم به زیاد مرتب شده‌اند. این ۷ گروه کاری خود به تعدادی زیر کار تقسیم شده‌اند. در جدول شماره ۱ مشخصات این برنامه‌ها و تعداد زیر کارهای هر کدام مشخص شده است. لازم به ذکر است برنامه‌های زیر که به عنوان ورودی به برنامه داده شده است از نظر حجم کاری قابل قیاس با NPB[۲۸] می‌باشند.

در جدول شماره ۲ نتایج اجرای کارهای درخواست شده بر روی مدل ارائه شده آمده است با این توضیح که این اجرا در پایین‌ترین سطح درخواست کاربر است. به عبارتی یک زمانبندی محلی است.

دومین تستی که انجام شده، مقایسه بین دو الگوریتم ژنتیک و لیست است با این توضیح که در این حالت نیز روش سلسله مراتبی اعمال شده است اما تفاوت

کار در اینجاست که درخواست کار در دو سطح بالایی مدل ارائه شده است. نتایج این تست در جدول شماره ۳ آمده است.

جدول ۱- تقسیم‌بندی کارهای وارد شده به سیستم

	تعداد زیر کارها	حجم برنامه (Gips)	متوسط حجم هر Task (Mips)
۱	۶	۳۳	۵۳۰۰
۲	۸	۳۹	۴۸۰۰
۳	۱۰	۶۹	۶۹۰۰
۴	۲۰	۱۷۰/۵	۸۵۰۰
۵	۵۰	۴۵۴/۵	۹۰۰۰
۶	۷۰	۶۲۵	۸۹۰۰
۷	۱۰۰	۹۰۹	۹۰۰۰

جدول ۲- زمانبندی اجرای کارها بطریق محلی

تعداد کارها	نود شماره ۱ (زمانبندی لیست)	نود شماره ۲ (زمانبندی لیست)	نود شماره ۳ (زمانبندی لیست)	نود شماره ۴ (زمانبندی لیست)
۶	۲۱۶	۴۴۷	۱۱۳۴	۱۹۳
۸	۳۳۶	۷۵۷	۱۹۹۹	۲۹۳
۱۰	۵۴۱	۱۳۵۲	۳۷۰۶	۴۶۲
۲۰	۱۹۰۴	۵۴۴۱	۱۵۵۸۴	۱۵۵۴
۵۰	۱۰۰۲۲	۳۱۱۳۱	۹۱۴۸۳	۷۹۳۳
۷۰	۱۸۹۰۵	۵۹۸۲۸	۱۷۶۷۶۰	۱۴۸۵۸
۱۰۰	۳۷۶۱۰	۱۲۰۸۱۴	۳۳۸۵۴۸	۲۹۳۸۱

همانطور که مشاهده می‌شود الگوریتم ژنتیک ارائه شده در قیاس با الگوریتم لیست، بهبود قابل ملاحظه‌ای را سبب شده است.

سومین تستی که انجام شده، مقایسه بین دو الگوریتم ژنتیک و Max-Min است. با این توضیح که در این حالت نیز روش سلسله مراتبی اعمال شده است اما تفاوت کار در اینجاست که درخواست کار در دو سطح بالایی مدل ارائه شده است. نتایج این تست در جدول شماره ۴ آمده است.

همانطور که مشاهده می‌شود الگوریتم ژنتیک ارائه شده در قیاس با الگوریتم Min-Max نیز، بهبود قابل ملاحظه‌ای را سبب شده است و این نشان می‌دهد که انتخاب الگوریتم ژنتیک بعنوان یک الگوریتم مرکزی، انتخاب مناسبی است چرا که با انتخاب بهترین مجموعه از منابع بهبود قابل ملاحظه‌ای را در زمان اجرا سبب می‌شود.

آخرین تستی که انجام گرفته است مقایسه بین عدم استفاده و استفاده از مدل سلسله مراتبی است. بعبارتی الگوریتم ژنتیک یکبار به عنوان زمانبند محلی و بار دیگر به عنوان زمانبند مرکزی استفاده شده است. نتایج این قیاس در جدول شماره ۵ آمده است.

جدول ۳- مقایسه زمان اجرای دو روش ژنتیک و لیست

تعداد کارها	GA	List	(List-GA)/GA
۶	۳۲۴/۲	۵۵۱/۲	۰/۷
۸	۴۴۹	۸۹۱	۰/۹۸
۱۰	۷۰۶	۱۵۸۳	۱/۲۴
۲۰	۴۶۳۵	۷۰۸۱/۷	۰/۵۲
۵۰	۲۸۱۳۳	۳۷۱۱۸/۹	۰/۳۱
۷۰	۶۱۰۸۷/۸	۶۳۸۲۶/۲	۰/۰۴
۱۰۰	۱۲۶۴۶۷/۳	۱۲۹۵۸۵/۸	۰/۰۲

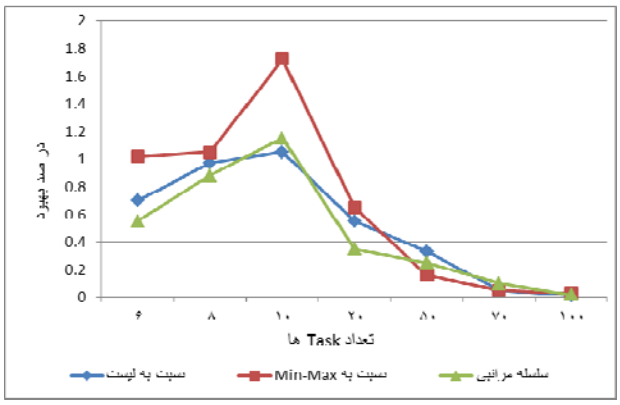
جدول ۴- مقایسه زمان اجرای دو روش ژنتیک و Max-Min

تعداد کارها	GA	Max-Min	(Max-Min-GA)/GA
۶	۳۲۴/۲	۶۳۵/۲	۱/۰۱
۸	۴۴۹	۹۳۰	۱/۰۷
۱۰	۷۰۶	۱۹۲۰	۱/۷۱
۲۰	۴۶۳۵	۷۶۵۰	۰/۶۵
۵۰	۲۸۱۳۳	۳۲۵۴۶	۰/۱۵
۷۰	۶۱۰۸۷/۸	۶۵۶۷۵/۴	۰/۰۷
۱۰۰	۱۲۶۴۶۷/۳	۱۳۲۱۴۱/۱	۰/۰۴

جدول ۵- مقایسه بین زمانبندی با روش ژنتیک با لحاظ کردن روش سلسله مراتبی و بدون لحاظ کردن روش سلسله مراتبی

تعداد کارها	Local	Central	(Local-Central)/Central
۶	۴۹۷/۵	۳۲۴/۲	۰/۵۳
۸	۸۴۶/۲۵	۴۴۹	۰/۸۸
۱۰	۱۵۱۵	۷۰۶	۱/۱۴
۲۰	۶۱۲۰	۴۶۳۵	۰/۳۲
۵۰	۳۵۱۴۲	۲۸۱۳۳	۰/۲۵
۷۰	۶۷۵۸۷/۷	۶۱۰۸۷/۸	۰/۱
۱۰۰	۱۳۱۵۸۸/۲	۱۲۶۴۶۷/۳	۰/۰۴

نتایج جداول ۳ و ۴ و ۵ در نمودار زیر آمده است.



شکل ۶- نمودار بهبود زمانبند به صورت ژنتیک در مقایسه با زمانبندی‌های لیست، Min-Max و همچنین با بهبود سلسله مراتبی

آنچه که از جداول ۱ تا ۵ قابل بررسی است آن است که الگوریتم ارائه شده نتایج مطلوبی را از نظر زمان اجرا سبب شده است. نکته قابل توجه در این است که الگوریتم ژنتیک به عنوان یک زمانبند مرکزی که به صورت مرکزی طراحی شده است عمل می‌کند و نتایج نشان می‌دهد که استفاده از این زمانبند مرکزی، زمان را در قیاس با زمانبند محلی سطح پایین‌تر بهبود می‌دهد، اما نکته قابل بحث در اینجاست که آیا استفاده از یک الگوریتم زمانبندی مرکزی بسیار دقیق که زمان اجرای طولانی دارد می‌تواند برای زمانبندی کارهای با حجم کم کارایی لازم را داشته باشد؟

در اجراهای مختلفی که بر روی تست این برنامه صورت گرفته است این نتیجه حاصل گردید که چنانچه تقسیم‌بندی بین کارها از نظر حجم عملیاتی صورت بگیرد و کارهای با حجمی متناسب به زمانبند مرکزی واگذار گردد هدف اصلی که همانا بالا بردن قدرت محاسباتی گرید است، بیشتر فراهم می‌گردد.

۶- نتیجه‌گیری

با هدف کاهش زمان اجرای یک برنامه کاربردی، استفاده از یک طرح سلسله مراتبی با سطوح مختلف تقاضای اجرای کار توسط کاربر، ما را تا حد زیادی به هدف نزدیک نمود. استفاده از دو الگوریتم زمانبندی ژنتیک و لیست به طور همزمان، باعث شد که قادر باشیم در مواردی بر اساس حجم کارو سطح درخواست کاربر از الگوریتم لیست به عنوان یک زمانبند محلی و از الگوریتم ژنتیک به عنوان زمانبند مرکزی‌ای که به صورت مرکزی عمل می‌کند استفاده نماییم. نتیجه استفاده از این روش تولید مناسبترین نگاشت بین کارها و منابع در دسترس است [۱۹ و ۲۰]. لازم به ذکر است که با این طبقه‌بندی مانع از این شدیم که کارهای با حجم کم وارد حلقه زمانبر اجرای الگوریتم ژنتیک شوند.

مراجع

[1] O. Beaumont, A. Legrand, and Y. Robert, "Optimal algorithms for scheduling divisible workloads on heterogeneous systems," *Proc. International Parallel and Distributed Processing Symposium*, 2003.

[2] E. S. H. Hou, N. Ansari, and H. Rong, "A Genetic Algorithm for Multiprocessor Schrduling," *IEEE Transactions on Parallel and Distributed Systems*, vol.5, no.2, pp.113-120, 1994.

[3] A. Andrieux, D. Berry, J. Garibaldi, S. Jarvis, J. MacLaren, D. Ouelhadj, and D. Snelling, *Open Issues in Grid Scheduling*, Technical Report, Edinburgh, UK, 2003.

[4] K. Czajkowski, I. T. Foster, N. T. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," *Proc. Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 62-82, 1998.

[5] J. M. Schopf, *A general architecture for scheduling on the Grid*, Argonne National Laboratory, 2002.

[6] H. Casanova, M. Kim, J. S. Plank, and J. J. Dongarra, "Adaptive scheduling for task farming with Grid middleware," *Journal of High Performance Computing Applications*. vol. 13, no. 3, pp. 231–240, 1999.

[7] V. Di Martino, "Sub Optimal Scheduling in Grid using Genetic Algorithms," *Proc. International Parallel and Distributed Processing Symposium*, pp., 22-26, 2003.

[8] F. Berman, R. Wolski, S. Figueria, J. Schopf, and G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks," *Proc. the ACM/IEEE Conference on Supercomputing*, Article 39, 1996.

[9] M. Wieczorek, R. Prodan, and T. Fahringer, "Scheduling of Scientific Workflows in the ASKALON Grid Environment," *ACM SIGMOD Record*, vol. 34, no. 3, pp. 56-62, 2005.

[10] S. S. Vadhiyar, and J. J. Dongarra, "A Metascheduler for Grid," *Proc. the 11th IEEE International Symposium on High Performance Distributed Computing*, pp. 343-351, 2002.

[11] P. Gradwell, *Grid Scheduling with Agents*, Technical Reoprt BA2 7AY, Department of Computer Science, University of Bath, UK, 2003.

[12] R. Bajaj, and D. P. Agrawal, "Improving Scheduling of Tasks in A Heterogeneous Environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 2, pp. 107–118, 2004.

[13] K. Aggarwal, and R. D. Kent, "An Adaptive Generalized Scheduler for Grid Applications," *Proc. The Annual International Symposium on High Performance Computing Systems and Applications*, pp. 15-18, 2005.

[14] Y. Gao, H. Rong, and J. Z. Huang, "Adaptve grid job scheduling with genetic algorithms," *Future Generation Computer System*, vol. 21, no. 1, pp. 151-161, 2004.

[15] M. Aggarwal, R. D. Kent, and A. Ngom, "Genetic Algorithm Based Scheduler for Computational Grids," *Proc. the Annual International Symposium on High Performance Computing Systems and Applications*, pp. 209-215, 2005.

[16] B. Nitzberg, and J. Schopf, "Current activities in the scheduling and resource management area of the global Grid forum," *Proc. the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 229-236, 2002.

[17] J. Lu, P. Ji, X. Wang, Y. Zhu, F. Li, T. Ma, and X. Wang, "Resource Management and Task Scheduling in Grid Computing," *Proc. the 8th International Conference on Computer Supported Cooperative Work in Design*, pp. 431-436, 2004.

[18] A. Y. Zomaya, "Genetic Scheduling for Parallel processor Systems: Comparative Studies and Performance Issues," *IEEE Transactions on Parallel and Distributed Systems*, vol.10, no.8, pp.795-812, 1999.

[19] F. Dong, and S. G. Akl, *Scheduling Algorithms for Grid Computing: State of the Art and Open Problems*, Technical Report No. 504, 2006.

[20] H. Topcuoglu, S. Hariri, and M. Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, 2002.

شیرین کریمی فارغ‌التحصیل رشته مهندسی کامپیوتر از دانشکده مهندسی برق و کامپیوتر دانشگاه شهید بهشتی و سپس همچنین دارای مدرک کارشناسی‌ارشد از دانشکده مهندسی کامپیوتر و فناوری اطلاعات در دانشگاه صنعتی امیرکبیر می‌باشد.

محمدکاظم اکبری فارغ‌التحصیل رشته مهندسی کامپیوتر از دانشگاه شهید بهشتی در سال ۱۹۸۴ و دارای مدرک کارشناسی‌ارشد و دکترا در رشته مهندسی کامپیوتر از دانشگاه کیس وسترن رزرو بترتیب در سال‌های ۱۹۹۱ و ۱۹۹۵ می‌باشد. او هم‌اکنون دانشیار دانشکده مهندسی کامپیوتر و فناوری اطلاعات در دانشگاه صنعتی امیرکبیر می‌باشد. زمینه‌های تحقیقاتی ایشان شامل شبکه‌های عصبی، طراحی سیستم، پردازش موازی و گرید و معماری کامپیوتر می‌باشد.
آدرس پست‌الکترونیکی ایشان عبارت است از:
akbari@ce.aut.ac.ir

سیدمحمدرضا میرزابابائی فارغ‌التحصیل مهندسی رشته مهندسی کامپیوتر با گرایش سخت‌افزار از دانشگاه صنعتی امیرکبیر در سال ۱۳۷۴ و کارشناسی ارشد هوش ماشین و رباتیک از دانشکده فنی دانشگاه تهران در سال ۱۳۷۷ می‌باشد. او هم‌اکنون دانشجوی دوره دکترای مهندسی کامپیوتر با گرایش معماری در دانشگاه صنعتی امیرکبیر می‌باشد.

اطلاعات بررسی مقاله:

تاریخ ارسال: ۸۵/۱۲/۱۰
تاریخ اصلاح: ۸۹/۴/۷
تاریخ قبول شدن: ۸۹/۵/۲۶
نویسنده مرتبط: دکتر محمدکاظم اکبری، دانشکده مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه صنعتی امیرکبیر، تهران، ایران.

---

[1] Heterogeneous
[2] Middleware
[3] Interface
[4] Throughput
[5] Task
[6] Job
[7] Time Slot
[8] Throughput
[9] Profit
[10] Interference
[11] Local Scheduler
[12] Superscheduler
[13] Metascheduler
[14] Load
[15] Type
[16] Shared Memory
[17] Distributed Memory
[18] Metascheduler
[19] Centralized
[20] Distributed
[21] High Performance Computing Grid
[22] Admin Domain
[23] Virtual Organization
[24] Search Space
[25] Predecessor
[26] Mutation
[27] Additive Mutation
[28] Nasa Parallel Benchmark