برنامه‌نویسی موازی مبتنی بر عامل‌های سیار بر روی گرید

روح‌الله مافی و حسین دلداری

چکیده: گرید[1]های محاسباتی بهره‌برداری از منابع توزیع‌شده محاسباتی را برای کاربردهایی که به محاسبات پرحجم نیاز دارند، فراهم می‌نمایند. توسعه برنامه‌هایی که قادر به استفاده از این امکانات باشند، یکی از چالش‌های پیش روی محاسبات گریدی می‌باشد. در این مقاله، با ارائه یک مدل برنامه‌نویسی موازی مبتنی بر عامل‌های سیار بر روی گرید، تلاشی می‌باشد که به منظور حل این مشکل صورت پذیرفته است. ارائه این مدل، که با توسعه بستر گریدی به نام Alchemi و افزودن خواص و نیز فرامین راهبری عامل‌ها به آن محقق گشته است، به کاربر اجازه می‌دهد تا با استفاده از حرکت عامل‌ها و ارتباط میان آن‌ها، برنامه موازی خود را توسعه دهد. این ایده از نوآوری‌های این مقاله محسوب می‌شود. به منظور ارزیابی این سیستم، الگوریتم ضرب ماتریس‌ها و نیز یافتن Convex Hull مجموعه‌ای از نقاط در سیستم مزبور پیاده‌سازی شده‌اند.

کلید واژه: **گرید، محاسبات گریدی، محاسبات توزیع‌شده، عامل‌های سیار، برنامه‌نویسی موازی، پردازش موازی، تحرک قوی، Alchemi.**

۱- مقدمه

گریدهای محاسباتی[2] [۱] و [۲] استفاده هماهنگ و قابل اطمینان از منابعی را که از نظر جغرافیایی توزیع شده‌اند، به منظور بهره‌برداری در کاربردهایی مانند محاسبات پرحجم، آنالیز داده‌های توزیع‌شده و تجسم از راه دور[3] میسر ساخته‌اند. منابع در گرید نامتجانس می‌باشند و این منابع ممکن است در حوزه‌های مدیریتی متفاوت و متعددی قرار داشته باشند و نیز نرم‌افزارهای متفاوتی را اجرا نمایند. این نرم‌افزارهای متفاوت موجب اعمال مقررات دسترسی متفاوت به آنها می‌گردد. همچنین شبکه‌هایی که گره‌های گرید را به هم وصل می‌نمایند ویژگی‌های کارکردی متنوعی دارند. بنابراین با توجه به موارد فوق توسعه و اصلاح کاربردها به منظور بهره‌گیری از محیط‌های گرید به صورت چالشی در برابر محاسبات گریدی در آمده است. هر یک از موارد فوق، انگیزه‌ای جهت انجام تحقیقات به منظور ارائه یک مدل برنامه‌نویسی توزیع‌شده برای استفاده در محیط‌های گرید فراهم می‌نماید. از جمله این تحقیقات می‌توان به گونه‌های متعدد سیستم‌های شیءگرا [۳] و [۴]، فن‌آوری‌های مبتنی بر وب [۱] و [۵]، محیط‌های حل مسائل [۶] و [۷]، CORBA [۸]، سیستم‌های گردش کاری[4]، سیستم‌های محاسباتی با توان بالا [۹] و [۱۰] و سیستم‌های مبتنی بر کامپایلرها [۱۱] اشاره نمود.

در این مقاله از عامل‌های سیار [۱۲] تا [۱۶]، به عنوان محمِلی برای برنامه‌نویسی موازی بر روی گرید استفاده شده است. قابلیت مهاجرتِ خودمختار عامل‌ها امکان استفاده از آن‌ها را در محیط شبکه‌ای که همواره در حال تغییر است، فراهم می‌نماید. حمل‌پذیری که از خواص ذاتی عامل‌ها می‌باشد (به علت پیاده‌سازی بر روی بسترهای مستقل از سکو مانند Java و .NET.)، امکان مدیریت شفاف توزیع وظایف در محیطی نامتجانس را فراهم نموده است. همچنین می‌توان از عامل‌ها در تعدیل بار کاری گره‌های شبکه نیز استفاده نمود. از مزایای اصلی استفاده از عامل‌های سیار در برنامه‌نویسی موازی، نسبت به استفاده از انتقال پیام، حرکت کد به سوی داده‌ها در کاربردهایی است که در آن‌ها حجم داده‌های مبادله‌شونده قابل توجه می‌باشد. به همین علت، عامل‌های سیار گزینه‌ای مناسب برای برنامه‌نویسی موازی، می‌باشند. همچنین برنامه توزیع‌شده‌ای که با استفاده از مدل برنامه‌نویسی عامل‌گرا نوشته شده باشد، خواناتر است و نیز مطابقت بیشتری با نگارش سِری آن دارد.

یکی از مشکلات برنامه‌نویسی موازی در محیط‌های انتقال پیام، کار با ساختارهای داده‌ای می‌باشد که دارای اشاره‌گر هستند (مانند درخت‌ها). زیرا در هنگام انتقال این نوع ساختار داده‌ها بین پردازش‌ها ساختار داده باید به صورت خطی درآید. ولی با استفاده از عامل‌های سیار به این کار نیازی نیست زیرا عامل‌های سیار کدی می‌باشند که به سمت داده حرکت می‌نمایند. در نتیجه لازم نیست که ساختار داده به صورت خطی درآید.

از سوی دیگر، بسترهای گرید کنونی، امکان اشتراک منابع محاسباتی را در وضعیتی مدیریت‌شونده و قابل اطمینان، برای کامپیوترهای شخصی و نیز ایستگاه‌های کاری فراهم نموده‌اند. این بسترها، محدودیت اتصال کامپیوترها از طریق یک LAN را از بین برده‌اند و با ایجاد توپولوژی سلسله مراتبی، حوزه بکارگیری از منابع را به شبکه‌های بسیار بزرگتری، گسترش داده‌اند (سیستم‌های عامل‌گرایی از قبیل Messengers [۱۷] و [۱۸] با این محدودیت مواجه می‌باشند). همچنین این بسترها، با اِعمال ملاحظات امنیتی، محیطی قابل اعتماد را در اختیار کاربران قرار می‌دهند.

در این مقاله، با انگیزه استفاده از امکانات گرید و نیز بهره‌مندی از مزایای عامل‌های سیار در ساخت برنامه‌های موازی از بستر گریدی با نام Alchemi [۱۹] که سیستمی کم‌حجم، پرقابلیت و با منبع باز می‌باشد، استفاده شده است. بستر گرید Alchemi از قابلیت تحرک ضعیف[5] عامل‌ها [۲۰] پشتیبانی می‌کند. قابلیت تحرک ضعیف اشیاء، یعنی زمانی که یک شیء انتقال می‌یابد (شیء به صورت نخ[6] پیاده‌سازی شده است)، فقط می‌توان آن را از نقطه آغازش اجرا نمود. در صورتی که در برنامه‌نویسی موازی توسط عامل‌های سیار، به قابلیت تحرک قوی[7] نیاز است. قابلیت تحرک را قوی گویند اگر امکان ادامه اجرای این نخ از نقطه‌ای که قبلاً تا آنجا اجرا شده است، میسر باشد. عیب دیگر این سیستم، این است که امکان ارتباط میان نخ‌ها در حین اجرا وجود ندارد. در نتیجه، سیستم Alchemi را فقط در مواردی می‌توان استفاده نمود که

این مقاله در تاریخ ۲۹ دی ماه ۱۳۸۳ دریافت و در تاریخ ۱۱ شهریور ماه ۱۳۸۵ بازنگری شد.

روح‌الله مافی، بخش مهندسی کامپیوتر، دانشکده فنی و مهندسی، دانشگاه فردوسی مشهد، مشهد ، ایران.

حسین دلداری، بخش مهندسی کامپیوتر، دانشکده فنی و مهندسی، دانشگاه فردوسی مشهد، مشهد ، ایران (email: hdeldari@yahoo.com).

1. Grid
2. Computational Grid
3. Remote Visualization
4. Workflow
5. Weak Mobility
6. Thread
7. Strong Mobility

کاربرد موازی به قطعات کاملاً مجزا از هم قابل تجزیه باشد. بنابراین از این سیستم برای موازی‌سازی سایر الگوهای الگوریتمی نمی‌توان استفاده نمود.

با توجه به نکات فوق، در این تحقیق، تغییرات بنیادینی در سیستم Alchemi داده شده است. یعنی امکان ارتباط میان‌نخی[1] و قابلیت تحرک قوی به آن افزوده گشته است. با ایجاد قابلیت اجرای فرامین راهبری[2] مربوط به عامل‌ها در این سیستم، امکان برنامه‌نویسی عامل‌گرا فراهم شده است.

سیستم جدید را، که پس از این تغییرات دارای قابلیت‌های افزون‌تری می‌باشد، +Alchemi نامیده‌ایم. همچنین آزمایشاتی که به منظور ارزیابی کارآیی سیستم فوق صورت گرفته است، بیانگر رسیدن به کارآیی بالاتر نسبت به سیستم MPICH-G۲ (نسخه‌ای از MPI که بر روی گرید گلوباس[3] قابل نصب است) می‌باشد.

در ادامه این مقاله نگاهی گذرا به پیش‌زمینه‌های مباحث به کار گرفته شده در این نوشتار خواهیم داشت و سپس با ذکر تغییرات اعمال‌شده در سیستم Alchemi، به ارزیابی کارآیی این سیستم توسط دو الگوریتم ضرب ماتریس‌ها و یافتن Convex Hull مجموعه‌ای از نقاط خواهیم پرداخت. در انتها با نتیجه‌گیری از کارهای انجام‌شده به کارهایی که در آینده در راستای این تحقیق قابل انجام می‌باشد اشاره خواهد شد.

۲- پیش‌زمینه

در این بخش به بیان پیش‌زمینه مختصری در مورد گرید، نقش عامل‌ها در محاسبات گریدی، سیستم Messengers و نیز سیستم Alchemi خواهیم پرداخت. سپس مختصری درباره مدل انتقال پیام MPI و پیاده‌سازی آن در گرید بحث خواهد شد. اگرچه MPI پیش‌زمینه‌ای بر این تحقیق نمی‌باشد ولی به علت اینکه زمان اجرای الگوریتم در این سیستم با زمان اجرای همان الگوریتم تحت MPI در گرید و نیز در سیستم انتقال پیام مقایسه شده است تا معیاری برای ارزیابی کارآیی وجود داشته باشد. بنابراین مختصری درباره آن توضیح داده شده است.

۲-۱ گرید

هدف از محاسبات گریدی، ترسیم یک کامپیوتر مجازی بزرگ، قدرتمند و خودمدیریت‌شونده است، که از اجتماع نامتقارنی از سیستم‌هایی که هر یک، ترکیبی از منابع را به اشتراک گذاشته‌اند، تشکیل شده است. در دهه گذشته، استانداردسازی ارتباطات میان سیستم‌های مختلف و نامتقارن موجب پدیدآمدن اینترنت شد و اکنون استانداردسازی اشتراک منابع با استفاده از پهنای باند بالا میان این سیستم‌ها، بزرگترین قدم در محاسبات گریدی است. گرید توسط یان فاستر[4] از بنیان‌گذاران پروژه گلوباس و مبدع اصطلاح گرید، به صورت زیر تشریح شده است:

گرید و فناوری‌های مرتبط در صدد تحقق اشتراک منابع به صورت کنترل‌شده و انعطاف‌پذیر در اندازه‌های بالا می‌باشند. این مهم به وسیله ساخت و توسعه پروتکل‌ها، سرویس‌ها و بسته‌های توسعه نرم‌افزار میسر می‌شود [۱].

در دهه گذشته پروژه‌هایی مانند FAFNER [۲۱] و I-WAY [۲۲] که تحت عنوان ابرمحاسبه‌گری[5] انجام شدند، زمینه را برای پروژه‌ها و بسترهای گرید کنونی فراهم آوردند. در حال حاضر بسترهای گرید فراوانی توسعه یافته‌اند که هر یک بر جنبه خاصی از محاسبات گریدی متمرکز شده‌اند. گلوباس [۲۳] که از مشهورترین بسترهای گرید می‌باشد بر پایه استاندارد معماری باز سرویس‌های گرید[6] (OGSA) بنا شده است.

۲-۲ عامل‌ها و محاسبات گریدی

تلاش‌های زیادی به منظور استفاده از عامل‌ها در جنبه‌های مختلفی از محاسبات گریدی صورت گرفته است. بخش عمده‌ای از این تلاش‌ها معطوف به ساخت گریدی با استفاده از عامل‌های سیار می‌باشد. در [۲۴] معماری چهارلایه‌ای به منظور ساخت یک گرید محاسباتی ارائه شده است که در لایه زیرین خود از عامل‌های سیار به عنوان محملی برای اشتراک منابع محاسباتی کامپیوترها استفاده می‌کند. فعالیت مشابهی در [۲۵]، به منظور ساخت گرید بر پایه عامل‌ها انجام شده است. در این کار، عامل‌ها از طریق یک "فضای چندتایی[7]" با یکدیگر در ارتباط می‌باشند. ماشین‌هایی که منابعی را در اختیار گرید می‌گذارند، به صورت یک عامل سیار بازنمایی می‌شوند. در پروژه A-Peer [۲۶] بستری بر روی JXTA[8] و با استفاده از سیستم IBM Aglets، به منظور محاسبات همتا-به-همتا[9] فراهم شده است. در [۲۷] به منظور حل مشکل بازنمایی کاربران سیار (از جمله برنامه‌هایی که بر روی تلفن همراه و یا کامپیوترهای همراه اجرا می‌شوند) از عامل‌های سیار در نمایه‌سازی[10] آنها بر روی بستر گریدی استاندارد (گلوباس)، استفاده شده است.

از عامل‌های سیار در ساخت گریدهای خاص‌منظوره[11] استفاده می‌شود. MyGrid [۲۸] مثالی از این نوع است که مورد استفاده بیولوژیست‌ها قرار می‌گیرد. در این پروژه، عامل‌ها در محاوره با کاربر (به منظور سفارشی‌کردن داده‌ها) و نیز در مذاکراتی که برای رسیدن به "سطح سرویس توافقی[12] (SLA)" انجام می‌شود، به کار گرفته شده‌اند. در پروژه مشابهی [۲۹]، از گریدی بر پایه عامل‌ها در مدیریت شبکه‌های مخابراتی و کامپیوتری استفاده شده است.

در تلاش‌های جداگانه دیگری، از عامل‌ها به منظور بهبود کارآیی نرم‌افزارهای موازی و توزیع‌شده [۳۰]، در بهینه‌سازی جستجو در گریدهای داده‌ای[13] [۳۱]، مدیریت منابع در گرید [۲۹]، بهبود تحمل خرابی در گرید [۳۲] و نیز مدیریت سرویس‌ها در گرید [۳۳] و [۳۴] استفاده شده است.

در مقاله حاضر، به شیوه‌ای متفاوت از تحقیقات فوق‌الذکر، از عامل‌های سیار به منظور برنامه‌نویسی بر روی بستر گرید استفاده شده است. این رویکرد موجب شده است که مزایای استفاده از عامل‌ها در برنامه‌نویسی موازی و نیز به کارگیری بستر گرید که فراهم‌کننده منابع محاسباتی فراوانی است، در اختیار کاربرد موازی قرار گیرد.

1. Inter-Thread Communication
2. Navigational Commands
3. Globus Toolkit
4. Ian T. Foster
5. Metacomputing
6. Open Grid Services Architecture
7. Tuple Space
8. A Sun Product for Peer-to-Peer Computing Stands for JuXTApose
9. Peer-to-Peer Computing
10. Profiling
11. Special Purpose
12. Service Level Agreement
13. Data Grid

۲-۳ سیستم Messengers

از مهم‌ترین سیستم‌هایی که از عامل‌های سیار در ساخت برنامه‌های موازی همه‌منظوره استفاده نموده‌اند، می‌توان به سیستم Messengers [۱۷] و [۳۵] تا [۳۷] که در دانشگاه UCI توسعه یافته است، اشاره نمود.

در این سیستم، کاربردها با استفاده از مجموعه‌ای از نخ‌ها ایجاد می‌گردد. این نخ‌ها قابلیت مهاجرت داشته و Messenger نامیده می‌شوند. مانند تمامی سیستم‌هایی که از قابلیت تحرک قوی پشتیبانی می‌کنند، هر Messenger، می‌تواند اجرای خود را متوقف ساخته، مقادیر متغیرهای خود را درون خود ذخیره کرده و به گره دیگری حرکت نماید. سپس در آن گره حالت خود را بازیابی می‌کند و اجرا را از محل قبلی ادامه می‌دهد. در این سیستم اعمال فوق‌الذکر توسط دستور ()hop انجام می‌شود.

در زبان معرفی‌شده توسط این سیستم، دو نوع متغیر وجود دارد: متغیرهای عامل و متغیرهای گره. متغیرهای عامل متعلق به هر عامل بوده و در ضمن حرکت عامل در شبکه، به همراه آن حمل می‌شود. ولی متغیرهای گره برای هر گره تعریف شده و توسط تمامی Messengerهایی که بر روی آن گره خاص قرار دارند، قابل دسترسی می‌باشد. در نتیجه، متغیرهای عامل می‌توانند به منظور حمل داده میان گره‌ها مورد استفاده قرار گیرند، در حالی که متغیرهای گره، توانایی مقادیر زیادی داده را بر روی خود داشته و می‌توانند در ارتباطات میان‌نخی مورد بهره‌برداری قرار گیرند. هر Messenger می‌تواند با استفاده از فرمان inject() عامل‌های دیگر را اجرا نماید. همگامی میان عامل‌ها توسط رویداد[1]ها صورت می‌پذیرد. با فرمان signalEvent() می‌توان یک عامل را بیدار کرد و با استفاده از فرمان waitEvent() یک عامل را می‌توان بلوکه نمود. بنابراین این فرمان‌ها مفاهیم بلوکه‌شدن[2] و بیدارشدن[3] را پیاده‌سازی می‌کنند. به جهت اینکه رویدادها محلی هستند، همگام‌سازی نیز در سطح یک گره انجام می‌شود و عامل‌ها مجاز نیستند که به داده‌های یکدیگر از راه دور دسترسی داشته باشند. برای انتقال کنترل از یک عامل به عامل دیگر، صراحتاً بایستی از دستورات ()hop و یا inject() استفاده نمود. این ویژگی "زمان‌بندی غیر قبضه‌ای[4]" نامیده می‌شود.

در سیستم Messengers از مفهوم عامل‌های سیار، بر خلاف بسیاری از سیستم‌های عامل‌گرای دیگر که عمدتاً مبتنی بر Java هستند، به عنوان یک مدل برنامه‌نویسی موازی استفاده شده است. قابلیت تحرک قوی موجود در سیستم Messengers، به این معنی است که محاسبات، و نه کُد، در طول شبکه حرکت می‌کند. به همین منظور در تبدیل اسکریپت Messengers به کد اجرایی، از کامپایلی دومرحله‌ای استفاده شده است. در مرحله اول برنامه Messengers، در محل عبارات راهبری، به توابع کوچکتری به زبان C شکسته می‌شود. اجرای این توابع توسط یک شمارنده برنامه منطقی با نام "اشاره‌گر به تابع بعدی" دنبال می‌شود [۳۵]. (اشاره‌گر در نام شمارنده برنامه با اغماض به کار رفته و به منظور اشاره به فضای آدرس مشخصی نمی‌باشد، این شمارنده دارای ساختمان داده ویژه‌ای می‌باشد و برای ذخیره‌سازی دستور اجرایی بعدی به کار می‌رود) این شمارنده برنامه منطقی و نیز متغیرهای عامل، از طریق انتقال پیام با استفاده از سوکت‌ها، بین گره‌ها رد و بدل می‌شوند. سربار این عمل (هر بلوک کنترلی Messenger[5] (MCB)، شامل اطلاعات داخلی در مورد عامل است) برای هر عامل، حدود ۲۰۰ بایت می‌باشد. در مرحله دوم؛ توابع C به زبان ماشین ترجمه شده و به صورت کتابخانه‌هایی به منظور اجرا، بر روی گره‌های شبکه بارگذاری می‌شوند.

در تحقیق دیگری با نام Jmessengers [۳۸] سیستم پیشنهادی Messengers بر روی جاوا پیاده‌سازی شده است که محصول به دست آمده از نظر کارآیی نسبت به Messengers ضعیف‌تر می‌باشد ولی از قابلیت حمل‌پذیری بیشتری برخوردار است.

در این مقاله نیز به منظور افزودن امکانات برنامه‌نویسی موازی مبتنی بر عامل‌های سیار بر روی گرید (سیستم Alchemi) از مدل پیشنهادی Messengers استفاده شده است با این تفاوت که سیستم توسعه‌یافته در این مقاله (Alchemi+) نسبت به Messengers از مزایایی از جمله مقیاس‌پذیری[6] بهتر، امنیت بالاتر و پیش‌پردازش سبک‌تر برخوردار است.

۲-۴ سیستم Alchemi

یکی از سیستم‌هایی که محاسبات گریدی، را بر روی شبکه‌ای از کامپیوترهای شخصی فراهم می‌نماید، سیستم Alchemi [۱۹] و [۳۹] می‌باشد. این سیستم برخلاف اغلب زیرساختارهای گرید، بر روی Windows و با استفاده از چارچوب NET. پیاده‌سازی شده است. این سیستم، علی‌رغم حجم کم، با برخورداری از معماری و مؤلفه‌های کارآمد، بستر انعطاف‌پذیر و مستحکمی را برای محاسبات گریدی فراهم آورده است.

در این سیستم، دو مدل نخ گریدی و کار گریدی، که به ترتیب در نوشتن برنامه‌های گریدی و اجرای برنامه‌های از قبل نوشته‌شده بر روی گرید کاربرد دارد پیشنهاد شده است. همچنین در این سیستم امکانی برای تعامل با دیگر میان‌افزارهای گریدی که بر اساس خدمات وب کار می‌کنند، نیز در نظر گرفته شده است. یکی از معایب این سیستم، عدم امکان ارتباط میان نخ‌های در حین اجرا می‌باشد. در نتیجه این نقصان، سیستم Alchemi تنها در مواردی که کاربرد موازی به قطعات کاملاً از هم مجزا تجزیه شود، کاربرد دارد و برنامه‌نویسی با آن در دیگر الگوهای ارتباطی مقدور نیست.

۲-۵ انتقال پیام

یک روش برنامه‌نویسی موازی رایج برای سیستم‌های با حافظه توزیع‌شده روش انتقال پیام می‌باشد. در این روش به زبان‌های رایج برنامه‌نویسی در کامپیوترهای سری تعدادی فرمان کتابخانه‌ای اضافه می‌گردد. زبان حاصل امکان برنامه‌نویسی موازی را فراهم می‌آورد. این کتابخانه‌ها شامل رویه‌هایی برای مقداردهی اولیه و پیکربندی محیط پیام‌ها مثل ارسال و دریافت بسته‌های داده می‌باشند. در حال حاضر دو سیستم انتقال پیام رایج MPI و PVM مورد استفاده قرار می‌گیرد که پیاده‌سازی‌های مختلفی از آنها وجود دارد. گرچه MPI پیش‌زمینه‌ای بر این تحقیق نمی‌باشد ولی به علت اینکه نتایج عملی اجرای الگوریتم‌های ضرب ماتریس و نیز Convex hull در سیستم Alchemi+ با نتایج همین الگوریتم‌ها در محیط MPI و نیز MPI در محیط گرید مقایسه شده است، بنابراین مبادرت به توضیح مختصری در مورد آنها می‌گردد. از کارهایی که در زمینه محاسبات موازی با استفاده از بسترهای گرید صورت

1. Event
2. Blocking
3. Wake up
4. Non-Preemptive Scheduling
5. Messenger Control Block
6. Scalability

جدول ۱: فهرست اعضاء کلاس GMonitor.

Enter, TryEnter	قفلی از یک شیء را به دست می‌آورد. این عمل می‌تواند به عنوان آغاز یک ناحیه بحرانی تلقی گردد. هیچ نخ دیگری نمی‌تواند وارد این ناحیه بحرانی شود، مگر آنکه دستورالعمل‌هایی در ناحیه بحرانی اجرا کند که از شیء قفل‌شده دیگری استفاده می‌کند.
Wait	قفل یک شیء را به منظور اجازه دسترسی و قفل‌کردن به دیگر نخ‌ها، آزاد می‌نماید. نخ فراخواننده این دستور تا زمانی که نخ دیگر یک شیء مورد نظر دسترسی دارد، منتظر می‌ماند.
Pulse (signal), PulseAll	سیگنالی به یک یا چند نخ در حال انتظار ارسال می‌کند. این سیگنال تغییر در وضعیت شیء قفل‌شده و نیز آمادگی برای آزادکردن قفل توسط ایجادکننده قفل را به نخ در حال انتظار اعلان می‌نماید. نخ در حال انتظار در صف آماده (object's ready queue) قرار گرفته، تا بتواند در زمان ممکن قفلی بر روی شیء به دست آورد.
Exit	قفل یک شیء را آزاد می‌کند. این عمل به عنوان نشانه‌ای بر انتهای ناحیه بحرانی که توسط شیء قفل‌شده، ایجاد شده بود، تلقی می‌گردد.

گرفته است می‌توان به MPICH-G۲ اشاره نمود. در این تحقیق ضمن ارائه معماری به کار گرفته شده در ساخت این میان‌افزار امکان اجرای برنامه‌های نوشته‌شده تحت MPI در محیط گرید میسر است. MPIAB [۴۰] یک مدل مبتنی بر عامل از MPI می‌باشد که از عامل‌ها در جاوا برای پیاده‌سازی توابع MPI استفاده نموده است. هدف این سیستم بهره‌گیری از تکنولوژی جاوا و عامل‌ها برای افزایش کارآیی ارتباطات و نیز همگامی بین گره‌ها بر روی شبکه می‌باشد.

۳- +ALCHEMI: محاسبات توزیع‌شده مبتنی بر عامل‌ها بر روی گرید

با توجه به نکات برشمرده‌شده در بخش ۲-۲ و برتری‌های برنامه‌نویسی موازی عامل‌گرا نسبت به برنامه‌نویسی موازی با استفاده از انتقال پیام، در این تحقیق تغییرات گسترده‌ای در سیستم Alchemi داده شده است. با استفاده از این تغییرات امکان برنامه‌نویسی موازی مبتنی بر عامل‌ها بر روی گرید فراهم شده است. از جمله امکاناتی که در این تحقیق به سیستم Alchemi افزوده شده است می‌توان از ایجاد ارتباط میان‌نخی و نیز ایجاد تحرک قوی نام برد.

۳-۱ ایجاد ارتباط میان‌نخی

در Alchemi، امکان ارتباط میان‌نخ‌ها (اشیاء کلاس GThread) پیاده‌سازی نشده است. به جهت نیاز به ارتباط میان‌نخی در برنامه‌نویسی مبتنی بر عامل‌های سیار (در کاربردهایی مانند همگام‌سازی) کلاس GMonitor، مشابه کلاس Monitor در چارچوب .NET. پیاده‌سازی شده است، با این تفاوت که کلاس GMonitor بر روی نخ‌هایی که بر روی کامپیوترهای مختلف قرار دارند، عمل می‌کند. در جدول ۱ فهرستی از متدهای کلاس GMonitor ارائه شده است.

در این سیستم با ایجاد یک قفل بر روی یک شیء با متد GMonitor.Enter، با استفاده از واسط هر عامل (که به صورت نخ پیاده‌سازی می‌شود) اطلاعات قفل و شیء قفل‌شده، به مدیر سیستم منتقل می‌شود. در گره مدیر سیستم، همواره فهرستی از قفل‌های اِعمال‌شده در سیستم کنونی موجود است. از این پس هرگاه عامل دیگری بخواهد با این عامل همگام گردد و یا اطلاعاتی را از این عامل دریافت نماید، از شیء مذکور استفاده می‌نماید و با متد GMonitor.EnterTry آن را وارسی می‌نماید. اگر قفلی بر روی آن عامل وجود داشته باشد تا زمانی که آن قفل آزاد نگردد، عامل درخواست‌کننده هماهنگی معلق می‌ماند (در فهرست قفل‌ها، شناسه عامل ایجادکننده قفل و عامل‌هایی که بر روی آن متوقف شده‌اند، نیز نگهداری می‌شود).

۳-۲ پیاده‌سازی قابلیت تحرک قوی

در سیستم +Alchemi عامل‌ها دارای تحرک قوی می‌باشند. در پیاده‌سازی قابلیت تحرک قوی برای عامل‌ها (که به صورت نخ پیاده‌سازی می‌شوند) کد دارای ویژگی قابلیت تحرک قوی به کد دارای ویژگی تحرک ضعیف ترجمه می‌گردد. در این مقاله مقصد ترجمه، کلاس GThread است که از کلاس‌های سیستم Alchemi بوده و دارای توانایی تحرک ضعیف می‌باشد.

هر متد، در کلاس اصلی عامل (GAgent) به یک کلاس درونی[1] Serializable (که در C# با نماد [Serializable] مشخص می‌شود) که حاوی رکورد فعال‌سازی آن متد است، ترجمه می‌شود. متغیرهای محلی، پارامترهای متد و شمارنده برنامه به فیلدهای این کلاس تبدیل می‌شوند. کلاس درونی شامل یک متد ()run است، که در برگیرنده بدنه اصلی متد می‌باشد. کلاس درونی عاملی که قابلیت تحرک قوی دارد، شامل آرایه‌ای از رکوردهای فعال‌سازی شیءهای کلاس اصلی است، که به صورت یک جدول مجازی از متدها، عمل می‌کند.

اجرای عبارات و نیز تغییر شمارنده برنامه، بایستی به صورت اتمیک انجام شوند، تا به عامل اجازه داده شود از مکانی به مکان دیگر، بدون از دست‌دادن اطلاعات، حرکت نماید. بدین علت کد اصلی به گونه‌ای ترجمه می‌شود که امکان ذخیره وضعیت عامل در هر لحظه میسر باشد. این عمل به نحوی انجام می‌شود که معانی عبارات حفظ شود و منطق برنامه نیز تغییر نکند. وجود قوانین ترجمه متفاوت برای عبارات مختلف در C# الزامی است.

متد ()Go، موجب حرکت یک عامل به مقصد جدیدش خواهد شد. لایه‌ای که توسط ویژگی [Serializable] مشخص شده است، وضعیت ایستای عامل را ذخیره می‌نماید. این لایه پس از رسیدن به مقصد، اشیائی از کلاس GThread ساخته و آن‌ها را به وضعیت اجرای قبلی‌شان باز می‌گرداند. عملگرهایی که اجرای آنها تا زمان معینی به طول خواهد انجامید، (مانند ()Thread.Sleep) متوقف شده و زمان باقیمانده از اجرای آنها، قبل از حرکت ذخیره می‌شود. در پیاده‌سازی انجام‌شده از قابلیت تحرک قوی، واسط IAgent و دو کلاس GAgent و ContextInfo وجود دارند.

۳-۲-۱ واسط IAgent

هر عامل سیار بایستی (مستقیم و یا غیر مستقیم) واسط IAgent را پیاده‌سازی کند. واسط IAgent، یک واسط نشانگر[2] است که لزوم پیاده‌سازی توابع این کلاس را توسط برنامه‌نویس به کامپایلر (و یا پیش‌پردازشگر) گوشزد می‌کند. تابع ()Go، عامل را به مقصد که توسط متغیری از کلاس GNode معین می‌شود، حرکت می‌دهد. واسط IAgent

1. Inner
2. Marker Interface

جدول ۲: فهرست اعضاء کلاس GAgent.

agentManager	اشاره‌گری به عامل است که Alchemi+Manager برای کنترل عامل از آن استفاده می نماید.
From	گره‌ای را که عامل قبل از حرکت در آنجا قرار دارد برمی‌گرداند.
FromLink	آخرین اتصالی را که عامل در آخرین حرکت خود طی کرده است برمی‌گرداند.
getContextInfo	اطلاعاتی را که عامل در بردارد برمی‌گرداند.
getNode()	گره‌ای را که هم‌اکنون عامل بر روی آن قرار دارد برمی‌گرداند.
Go	عامل را به گره دیگری حرکت می‌دهد.
Id	شناسه یکّه عامل را برمی‌گرداند.
Initialize	توسط +Alchemi برای مقداردهی اولیه عامل استفاده می‌شود.
Replicate(GNode)	نسخه‌ای از عامل جاری می‌سازد (با وضعیت مشابه) و آن را به مقصدی که توسط شیءای از کلاس GNode مشخص می‌شود حرکت می‌دهد.
Start	عامل فعالیت اصلی خود را از این متد آغاز می‌کند.کد محتوی برنامه کاربر در این متد قرار می‌گیرد.
Terminate	عامل جاری را از بین می‌برد.

به صورت زیر تعریف می‌شود:

```
public interface IAgent: MarshalByRefObject {
    public void Go (GNode dest) {...}
    public void Go (GNodeCollection dests) {...}
}
```

۳-۲-۲ کلاس GAgent

این کلاس، واسط IAgent را پیاده‌سازی کرده و متدهای متعددی علاوه بر ()Go را فراهم می‌آورد. متد ()getContextInfo از اعضاء این کلاس، اطلاعاتی در مورد محتویاتی که در حال اجرای آنها می‌باشد، در اختیار می‌گذارد. در جدول ۲ فهرستی از متدهای مهم کلاس GAgent ارائه شده است.

۳-۲-۳ کلاس ContextInfo

از این کلاس، هر عامل برای دسترسی به منابعی از ماشین که این عامل بر روی آن اجرا می‌شود استفاده می‌نماید. این منابع، شامل اشیاء متعلق به سیستم می‌باشند که میزبان می‌خواهد آنها را در اختیار عامل سیار بگذارد. در حال حاضر، در این مقاله متد ()getThreadIdentifier برای این کلاس پیاده‌سازی شده است. این متد شناسه نخی را که در حال اجرای عامل می‌باشد برمی‌گرداند.

۳-۲-٤ محل قرارگیری کد کاربر

کد کاربر بایستی در کلاسی که از کلاس GAgent مشتق شده است، نوشته شود. در این کلاس می‌توان متدهای دیگری نیز تعریف و پیاده‌سازی نمود. اجرای عامل از متد ()void Start آغاز می‌شود و بنابراین کد اجرایی کاربر بایستی در این متد قرار گیرد. کدی که در این متد نوشته می‌شود، از متدهای کلاس GAgent (کلاس پدر) به منظور نوشتن الگوریتم‌های موازی عامل‌گرا استفاده می‌کند. همچنین در این

```
public void sample (int x) {
    int y;
    // blocks of statements
    BC1
    BC2
}
```

شکل ۱: ساختار ساده یک متد (sample).

```
[Serializable]
protected class _sample: MarshalbyRefObject {
    int x, y, progCounter=0; Object trgt;
    void setPCForMove() {...}
    void run() {...}
    Try {...
        this.request_read();
        if ((progCounter==0)) {
            progCounter+=1; BC1 }
        this.read_accomplished();
        this.request_read();
        if ((progCounter==1)) {
            progCounter+=1; BC2 }
        this.read_accomplished(); }
    catch(Exception e) {...}}...
}
```

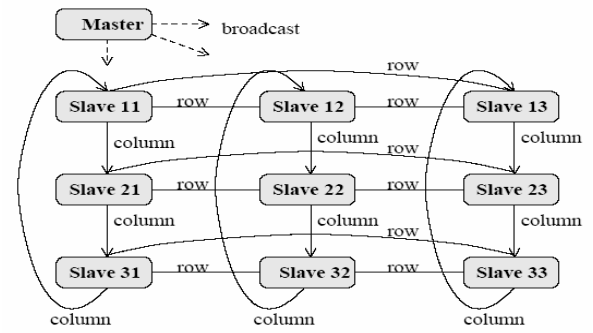
شکل ۲: کلاس تولیدشده از متد sample.

کلاس بایستی متد سازنده[1] نیز تعریف گردد. فراخوانی این کلاس و مقداردهی اولیه آن، مانند دیگر برنامه‌های Alchemi می‌باشد.

۳-۲-۵ ترجمه متدها

پیش‌پردازشگر برای هر متد عامل، یک کلاس که اشیاء آن، رکوردهای فعال‌سازی آن متد را ارائه می‌دهند، تولید می‌نماید. کلاس رکورد فعال‌سازی هر متد، به صورت زیر کلاسی از کلاس MarshalByRefObject (که برای اجرای کد بر روی مقصد راه دور الزامی است) تعریف می‌شود. تابع sample را به صورتی که در شکل ۱ آمده است، در نظر بگیرید. پارامتر x، متغیر محلی y و شمارنده برنامه، فیلدهایی از کلاس _sample خواهند بود. وجود متدی مانند ()setPCforMove به منظور تعلیق اجرا و حرکت‌دادن یک نخ، در کلاس تولیدشده الزامی است. این متد شمارنده برنامه جاری را ذخیره کرده و سپس مقدار ۱- را به آن می‌دهد. این کار به منظور اطمینان از اینکه هیچ دستوری قبل از حرکت عامل ناتمام باقی نمانده است، صورت می‌پذیرد. متد ()run شامل نگارش ترجمه‌شده بدنه تابع ()sample خواهد بود. در این تابع، کدی که شمارنده برنامه را می‌افزاید و نیز کدی که اجازه می‌دهد تابع ()run پس از حرکت از وضعیت قبلی اجرا گردد قرار دارد. هر نخ برای از بین‌بردن مشکلات همگام‌سازی، قبل و بعد از اجرای هر عبارت به ترتیب قفلی را تقاضا و آزاد می‌کند. این کار توسط توابع ()request_read و ()read_accomplished انجام می‌گردد. کلاس رکورد فعال‌سازی تولیدشده از تابع ()sample به صورت شکل ۲ است.

1. Constructor



شکل ۳: مسئله ضرب ماتریس‌ها: مثالی از الگوریتم‌های محاسبات عددی.

جدول ۳: مشخصات محیط مورد استفاده در بررسی کارآیی الگوریتم‌ها.

	MPICH	MPICH-G۲	Alchemi+
پردازشگرها	۸×P۴ ۲٫۴ GHz	۱×P۴ ۲٫۴ GHz	۸×P۴ ۲٫۴ GHz
سیستم عامل	Redhat Linux ۹٫۰	Redhat Linux ۹٫۰	Windows XP
میان‌افزار	OS Kernel	Globus Toolkit ۳٫۲	.NET Framework

۳-۳ اصلاحات تکمیلی

علاوه بر تغییرات ذکرشده در قسمت‌های قبلی، تغییرات زیادی در اغلب کلاس‌های سیستم Alchemi به منظور تطبیق آن با مدل برنامه‌نویسی موازی مبتنی بر عامل‌های سیار صورت گرفته است. از جمله این تغییرات می‌توان به اصلاحات در کلاس‌های GNode، GThread، GApplication و GManager و نیز تعریف کلاس GLink اشاره نمود.

٤- ارزیابی کارآیی

پس از اصلاحات انجام‌شده و افزودن کلاس‌هایی که برنامه‌نویسی عامل‌گرا را در Alchemi میسر می‌نمایند، به منظور بررسی کارآیی این سیستم، دو الگوریتم "یافتن Convex Hull مجموعه‌ای از نقاط" و "ضرب ماتریس‌ها" به صورت عامل‌گرا روی این سیستم (Alchemi+) پیاده‌سازی شده است. این دو الگوریتم با استفاده از انتقال پیام بر روی دو محیط MPICH و MPICH-G۲ (برنامه‌نویسی مبتنی بر MPI بر روی گرید گلوباس) نیز پیاده‌سازی شده است. سپس نتایج به دست آمده از اجرای این الگوریتم‌ها بر روی محیط Alchemi+ با نتایج به دست آمده از اجرای آنها بر روی MPICH و MPICH-G۲ مقایسه شده است. محیط اجرای الگوریتم‌ها در جدول ۳ ارائه گشته است. در ادامه، نحوه پیاده‌سازی این دو الگوریتم و بررسی کارآیی آن‌ها به تفکیک خواهد آمد.

٤-۱ ضرب ماتریس‌ها

الگوریتم ضرب ماتریس‌ها، یکی از الگوریتم‌های پرکاربرد در محاسبات عددی می‌باشد، به روش‌های مختلفی در محیط‌های توزیع‌شده قابل پیاده‌سازی می‌باشد. یکی از روش‌های ضرب ماتریس‌ها، به صورت چرخشی بلوکی می‌باشد. شکل ۳ نحوه انجام این الگوریتم را نمایش می‌دهد. در شبه‌کد این الگوریتم که با استفاده از Alchemi+API نوشته شده و مبتنی بر عامل‌های سیار می‌باشد شکل ۴، وجود یک شبکه منطقی از کامپیوترها بر روی گرید مفروض در نظر گرفته شده است. عاملی که در گره master قرار دارد، خود را بر روی تمامی گره‌های slave منتشر می‌نماید (خط ۸-۹).

```
1  matrix_mult(m) {
2      master=this.getContextInfo();
3      procs=m*m;
4      for (k=0; k<m; k++) {
5          sync=0;     /*go to all slave nodes*/
6          for (i=0; i<GNodeCollection.getNodeCount(); i++)
7              this.Replicate(GNodeCollection.Item(i));
8          if (node.j==(node.i+k)%m) {
9              //multicast A to all nodes in the row
10             this.A=copy(this.getNode().A);
11             for (i=0; i<GNodeCollection.getNodeCount(); i++)
12                 if (GNodeCollection.Item(i).getName().equals("row")
13                     this.Replicate(GNodeCollection.Item(i));
14         } this.Terminate();
15         multiply();     //Cij=Aij*Bij
16         try {     //barrier synchronization
17             GMonitor.Enter(sync);
18             sync++;
19             if (sync!=proc) GMonitor.Wait(sync);
20             else GMonitor.PalseAll(sync);
21         }
22         finally {
23             sync=0;
24             GMonitor.Exit(sync); }
25         this.B=copy(this.getNode().B);
26         //rotate B to column i-1
27         this.Go(this.FromLink("-column"));
28         this.getNode().B=copy(this.B);
29         try {     //barrier synchronization
30             GMonitor.Enter(sync);
31             sync++;
32             if (sync!=proc) GMonitor.Wait(sync);
33             else GMonitor.PalseAll(sync);
34         }
35         finally {
36             sync=0;
37             GMonitor.Exit(sync);
38         }
39     }
40 }
```

شکل ۴: حل مسئله ضرب ماتریس‌ها در سیستم Alchemi+.

عامل‌هایی که بر روی قطر قرار دارند، به همراه قسمتی از ماتریس **A** که در اختیار آنها می‌باشد، به دیگر گره‌های slave در آن سطر حرکت می‌کنند (خط ۱۰-۱۶) آنگاه کلیه عامل‌ها **A**×**B** را محاسبه کرده و سپس فرآیند همگام‌سازی انجام می‌شود (خط ۱۷-۲۷). عامل‌ها سپس، قسمتی از ماتریس **B** که در اختیار دارند دوران داده (خط ۲۸-۳۱) و مجدداً همگام‌سازی انجام می‌گردد (خط ۳۲-۴۱).

نمودارهای کارآیی حاصل از اجرای این الگوریتم بر روی سه محیطی که در جدول ۳ برشمرده شده‌اند، در شکل ۵-الف و ۵-ب به ترتیب به ازای ضرب ماتریس‌های [۱۶۰,۱۶۰]×[۱۶۰,۱۶۰] و [۱۸۰,۱۸۰]×[۱۸۰,۱۸۰] نمایش داده شده‌اند.

جدول ۴: زمان اجرای الگوریتم ضرب ماتریس‌ها در سه محیط.

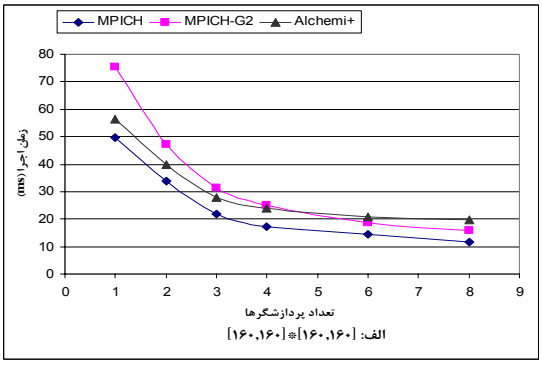
الف: [۱۶۰,۱۶۰]×[۱۶۰,۱۶۰]			
تعداد بستر اجرا / پردازشگرها	MPICH	MPICH-G۲	Alchemi+
۱	۴۹٫۶۵۵	۷۵٫۳۵۱	۵۶٫۵۰۴
۲	۳۳٫۷۵۲	۴۷٫۴۹۲	۴۰٫۶۹۸
۳	۲۱٫۷۶۴	۳۱٫۷۱۲	۲۸٫۵۳۷
۴	۱۷٫۳۱۱	۲۴٫۸۹۲	۲۴٫۲۶۷
۶	۱۴٫۲۰۳	۱۸٫۸۰۲	۲۰٫۹۱۴
۸	۱۱٫۶۸۳	۱۵٫۸۹۶	۱۹٫۸۰۰
ب: [۱۸۰,۱۸۰]×[۱۸۰,۱۸۰]			
تعداد بستر اجرا / پردازشگرها	MPICH	MPICH-G۲	Alchemi+
۱	۷۶٫۳۸۰	۱۰۳٫۰۶۴	۸۹٫۷۳۷
۲	۵۲٫۱۵۸	۷۳٫۷۳۶	۶۵٫۴۲۱
۳	۳۹٫۱۱۹	۵۴٫۶۰۹	۴۸٫۰۰۰
۴	۲۹٫۰۳۵	۴۰٫۲۸۶	۳۶٫۰۵۰
۶	۲۰٫۶۸۹	۲۸٫۱۱۹	۲۷٫۱۴۰
۸	۱۷٫۱۰۰	۲۱٫۴۰۷	۲۴٫۴۱۲

نتایج به دست آمده از این آزمایش‌ها در جدول ۴ نمایش داده شده است. واحد زمان اجرای برنامه‌ها میلی‌ثانیه می‌باشد.

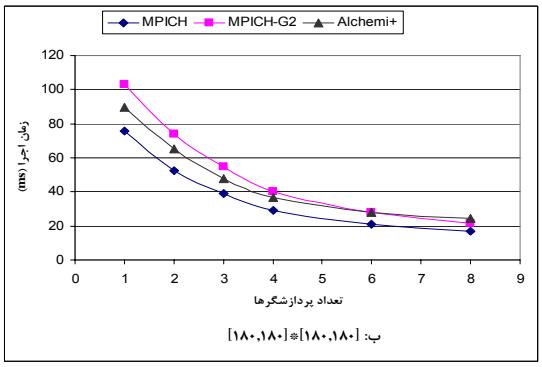
دلیل اینکه کارآیی الگوریتمی که با استفاده از سیستم Alchemi+ نوشته شده است، با افزایش تعداد پردازشگرها کاهش می‌یابد و نسبت به برنامه اجراشده بر روی MPICH و MPICH-G۲ ضعیف‌تر عمل می‌کند، این است که با افزایش تعداد پردازشگرها بلوک‌هایی که در هر پردازشگر در هر تکرار محاسبه می‌شوند، کوچک‌تر شده و در نتیجه نسبت ارتباطات به محاسبات افزایش می‌یابد و نیز به علت سرباری که این سیستم در زمان ارتباطات بر داده واقعی اضافه می‌کند که مجموع سربارهای تحمیلی .NET، Alchemi و نهایتاً Alchemi+ می‌باشد، سرعت اجرای برنامه کاهش می‌یابد، در صورتی که برنامه‌ای که بر روی MPICH اجرا می‌شود، با کمترین سربار ممکن اجرا می‌شود.

همان‌طوری که شکل ۵-ب نشان می‌دهد با افزایش ابعاد ماتریس کارآیی الگوریتم در سیستم Alchemi+ بهبود یافته و زمان اجرای آن به زمان اجرای این الگوریتم در محیط MPI نزدیک می‌شود. مزیت عمده این سیستم این است که امکان اجرای موازی بر مبنای عامل‌ها یک الگوریتم را بر روی یک گرید که به صورت جغرافیایی پراکنده است فراهم می‌نماید.

در حقیقت سربارهای اضافه‌شده به سیستم و در نتیجه افزایش زمان اجرای الگوریتم، هزینه‌ای است که کاربر در ازای وسیع‌ترشدن دامنه اجرای برنامه‌اش و امکان اجرای آن بر روی شبکه بزرگی که می‌تواند از LANهای متعددی ساخته شده باشد، می‌پردازد. البته به دلیل اینکه آزمایشی که بر روی MPICH-G۲ انجام شده تنها بر روی یک کامپیوتر صورت گرفته و نتایج این آزمایش به منظور مقایسه با دیگر آزمایش‌ها تسطیح شده‌اند، نمی‌توان کارآیی ضعیف‌تر Alchemi+ را نسبت به MPICH-G۲، مورد اطمینان قرار داد. در حالت کلی، به دلیل اینکه بسترهای گرید، مدعی استفاده از منابع بی‌کار کامپیوترهایی که از لحاظ جغرافیایی پراکنده و در LANهای مختلفی قرار دارند، می‌باشند، اجرای برنامه‌هایی بر روی گرید که مؤلفه‌های آنها با یکدیگر ارتباطات پرحجمی دارند از کارآیی خوبی برخوردار نیست و منطقی به نظر نمی‌رسد.

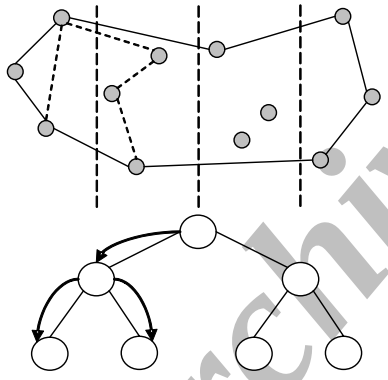


(الف)



(ب)

شکل ۵: نمودار کارآیی سه محیط در اجرای الگوریتم ضرب ماتریس‌ها با ابعاد (الف) [۱۶۰×۱۶۰] و (ب) [۱۸۰×۱۸۰].



شکل ۶: مسئله Convex Hull: مثالی از الگوریتم‌های تقسیم و حل.

۴-۲ یافتن Convex Hull مجموعه‌ای از نقاط

الگوریتم یافتن Convex Hull مجموعه‌ای از نقاط، که از نوع الگوریتم‌های تقسیم و حل به شمار می‌رود، نیازمند ساخت درخت جستجو به گونه‌ای پویا می‌باشد. نگارش داده‌ها بر روی این درخت، از نکات مهم در حل این مسئله می‌باشد. همان‌طور که در شکل ۶ دیده می‌شود، برنامه نقاط را به دو زیرمجموعه با اندازه $[n/۲]$ به نحوی که هر زیرمجموعه در یک طرف خط قرار گیرند، تقسیم می‌کند. سپس Convex Hull هر یک از مجموعه‌ها، به صورت بازگشتی با همین روش محاسبه می‌شود. حل‌های جزئی در هر سطح در فاز برگشت (Backtracking) با یکدیگر ادغام می‌شوند. آسان‌ترین راه برای پیاده‌سازی الگوریتم فوق به صورت سِری، استفاده از توابع بازگشتی در ساخت و جمع‌آوری نتایج از درخت می‌باشد. در نگارش موازی این الگوریتم، عمل‌بازگشت[1] به صورت ساخت

1. Recursion

```
1. convex_hull(points)
2. {
3.     for (i=0; i<max_level; i++) {
4.         this.Replicate(new GLink("right"));
5.         this.Replicate(new GLink("left"));
6.         mid_pt=top_pt+(tail_pt-top_pt)/2;
7.         if (this.FromLink().getName=="left")
8.             tail_pt=mid_pt-1;
9.         if (this.FromLink().getName=="right")
10.            top_pt=mid_pt-1;
11.    }
12.    my_pt=convex(top_pt, tail_pt);
13.    for (i=max_level; i>0; i--) {
14.        this.Go(this.From());
15.        if (other_pt==NULL) {
16.            other_pt=my_pt;
17.            this.Terminate();
18.        } else
19.            my_pt=merge(my_pt, other_pt);
20.    }
21. }
```

شکل ۷: حل مسئله Convex Hull در سیستم +Alchemi.

جدول ۵: زمان اجرای الگوریتم CONVEX HULL در سه محیط.

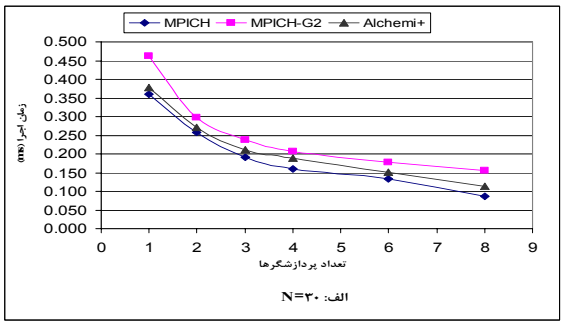
الف: $N=۳۰$			
تعداد بستر اجرا / پردازشگرها	MPICH	MPICH-G۲	Alchemi+
۱	۰٫۳۶۰	۰٫۴۶۲	۰٫۳۷۷
۲	۰٫۲۵۸	۰٫۲۹۷	۰٫۲۷۱
۳	۰٫۱۹۲	۰٫۲۳۸	۰٫۲۱۲
۴	۰٫۱۵۹	۰٫۲۰۷	۰٫۱۹۰
۶	۰٫۱۳۴	۱۷۸٫۰	۰٫۱۵۱
۸	۰٫۰۸۷	۰٫۱۵۵	۰٫۱۱۳
ب: $N=۵۰$			
تعداد بستر اجرا / پردازشگرها	MPICH	MPICH-G۲	Alchemi+
۱	۲٫۰۵۵	۲٫۳۶۳	۲٫۱۵۸
۲	۱٫۱۰۴	۱٫۴۳۰	۱٫۲۷۰
۳	۰٫۶۵۱	۰٫۹۸۴	۰٫۸۱۰
۴	۰٫۴۶۲	۰٫۷۸۷	۰٫۶۹۳
۶	۰٫۳۴۴	۰٫۵۵۱	۰٫۴۸۱
۸	۰٫۲۳۸	۰٫۵۰۳	۰٫۳۸۵

فرآیندهای جدید و انتقال داده به آن‌ها صورت می‌پذیرد.

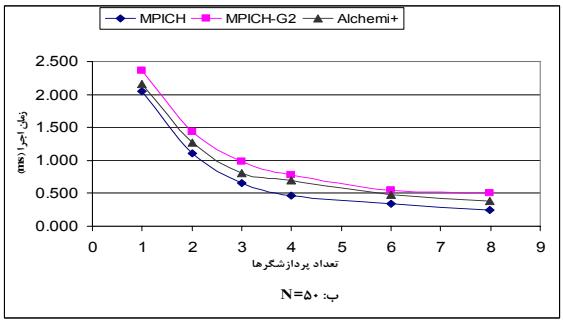
در شکل ۷ برنامه‌ای که با استفاده از Alchemi+API و با بهره‌گیری از عامل‌های سیار و خواص آن‌ها، برای حل این مسئله (با حذف قسمت‌های غیر ضروری)، ارائه شده است.

نمودارهای کارآیی حاصل از اجرای این الگوریتم بر روی سه محیطی که در جدول ۳ برشمرده شده‌اند، در شکل ۸-الف و ۸-ب به ترتیب به ازای ۳۰ و ۵۰ نقطه نمایش داده شده‌اند.

همچنین نتایج به دست آمده از این آزمایش‌ها در جدول ۵ نمایش داده شده است.



(الف)



(ب)

شکل ۸: نمودار کارآیی سه محیط در اجرای الگوریتم Convex Hull (الف) $N=۳۰$ و (ب) $N=۵۰$.

در این الگوریتم حجم داده‌های منتقل‌شده توسط عامل‌ها کمتر از الگوریتم ضرب ماتریس‌ها می‌باشد. همان‌طور که شکل ۸ نشان می‌دهد کارآیی الگوریتم Convex Hull در سیستم +Alchemi به کارآیی این الگوریتم در سیستم MPICH نزدیک‌تر و از کارآیی این الگوریتم در سیستم MPICH-G۲ بهتر می‌باشد. کارآیی پایین‌تر سیستم +Alchemi نسبت به MPICH را مانند مثال گذشته می‌توان در سربار چارچوب .NET نسبت به هسته لینوکس و سربارهای افزون‌تری که به جهت امکان قابلیت تحرک قوی به سیستم Alchemi اضافه شده است، جستجو کرد. بهینه‌سازی تبدیل کد قوی به کد ضعیف و تقسیم داده مناسب‌تر میان عامل‌ها می‌توانند به عنوان راه‌حل‌هایی برای این مشکل در نظر گرفته شوند.

۵- نتیجه‌گیری و کارهای آینده

در این مقاله سیستم +Alchemi که محیط برنامه‌نویسی موازی مبتنی بر عامل‌های سیار را بر روی گرید فراهم آورده است، ارائه گشته است. این کار با افزوده‌شدن دستورات پایه‌ای در برنامه‌نویسی موازی با استفاده از عامل‌ها، بر روی سیستم Alchemi که فراهم‌کننده بستر گرید بر روی کامپیوترهای شخصی می‌باشد، میسر گشته است. از مهم‌ترین اصلاحات در این سیستم، می‌توان به افزودن کلاس GAgent که برنامه‌های کاربر با ارث‌بری از آن ساخته می‌شوند و شامل دستورات راهبری و دیگر ملزومات برنامه‌نویسی عامل‌گرا می‌باشد و نیز کلاس GMonitor که امکان ارتباط میان نخ‌های در حال اجرا بر روی گرید را فراهم می‌سازد (که در Alchemi ممکن نبوده است)، اشاره داشت. در ادامه با اجرای دو الگوریتم ضرب ماتریس‌ها و یافتن Convex hull مجموعه‌ای از نقاط که به ترتیب به الگوریتم‌های محاسبات عددی و "تقسیم و حل" تعلق دارند، کارآیی این سیستم مورد بررسی قرار گرفته است. با بررسی نتایج می‌توان ادعا نمود که سیستم فوق در اغلب موارد بهتر از سیستم MPICH-G۲ عمل کرده است. از دلایل ضعف سیستم نسبت به MPICH می‌توان به

سربار بیشتر چارچوب NET. نسبت به هسته لینوکس اشاره داشت. علاوه بر سربار فوق، سربار دیگری نیز به منظور تبدیل کد با قابلیت تحرک قوی به کدی با تحرک ضعیف، که در Alchemi قابل اجرا باشد، به برنامه‌های Alchemi+ افزوده می‌شود. همچنین بهره سرعت[1] اندازه‌گیری‌شده برای دو الگوریتم فوق دارای رشد تقریباً خطی و قابل قبولی می‌باشد.

از نوآوری‌های این مقاله می‌توان به ایجاد قابلیت تحرک قوی در چارچوب NET. و نیز ارائه مدل برنامه‌نویسی موازی مبتنی بر عامل‌های سیار بر روی بستر گرید اشاره نمود. گرید توسعه‌یافته در این تحقیق Alchemi+ نام‌گذاری شده است که یک گرید بر پایه عامل‌ها می‌باشد.

از دیگر نوآوری‌های این مقاله، مقایسه‌ای است که میان سه محیط مطرح‌شده صورت پذیرفته است. بنا بر اطلاعات نگارندگان تاکنون مقایسه‌ای با این خصوصیات در تحقیق دیگری انجام نشده است.

از مجموعه کارهای قابل انجام در راستای این مقاله در آینده، می‌توان از اجرای الگوریتم‌های پرکاربرد دیگری از قبیل FFT[2]، BSP Tree[3] و CSG[4] بر روی سیستم Alchemi+ نام برد.

به منظور افزایش کارآیی سیستم می‌توان اصلاحاتی در نحوه همگام‌سازی عامل‌های سیار انجام داد.

همچنین می‌توان کلیه الگوریتم‌ها را از نظر طبیعتشان به چند دسته تقسیم نمود و برای هر دسته از الگوریتم‌ها یک اسکلت بر روی سیستم Alchemi+ به صورت کارا پیاده‌سازی نمود. در این صورت می‌توان کارآیی اجرای الگوریتم‌ها را به نحو چشمگیری بر روی سیستم Alchemi+ بهبود بخشید.

مراجع

[1] I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 1999.

[2] I. Foster, "The grid: a new infrastructure for 21st century science," *Physics Today*, vol. 54, no. 2, pp. 42-47, Feb. 2002.

[3] D. Gannon and A. Grimshaw, *Object-Based Approaches,* in *The Grid: Blueprint for a New Computing Infrastructure*, ed. by I. Foster and C. Kesselman, pp. 205-236, Morgan Kaufmann, 1998.

[4] A. S. Grimshaw and W. A. Wulfs, "The legion vision of a worldwide virtual computer," *Communications of the ACM*, vol 40, no. 1, pp. 39-45, Jan. 1997.

[5] A. Vahdate, E. Belani, P.l Eastham, C. Yoshikawa, T. Anderson, D. Culler, and M Dahlin, "WebOS: operating system services for wide area applications," in *Proc. 7th Symp. on High Performance Distributed Computing*, pp. 52-63, Jul. 1998.

[6] H. Casanova and J. Dongarra, *Netsolve: A Network Server for Solving Computational Science Problems*, Technical Report CS-95-313, University of Tennessee, Nov. 1995.

[7] H. Nakada, M. Sato, and S. Sekiguchi, "Design and implementations of Ninf: towards a global computing infrastructure," *Future Generation Computing Systems*, vol. 15, no. 5, pp.649-658, Oct. 1999.

[8] M. Henning and S. Vionski, *Advanced CORBA® Programming with C++*, Addison-Wesley Pub Co, 1st edition, 1999.

[9] D. Abramson, R. Sosic, J. Giddy, and B. Hall, "Nimrod: a tool for performing parameterized simulations using distributed workstations," in *Proc. 4th IEEE Sump. on High Performance Distributed Computing IEEE Computer Society Press*, pp. 112-121, 1995.

[10] M. Litzkow, M. Livny, and M. Mutka, "Condor: a hunter of idle workstations," in *Proc. 8th Intl Conf. on Distributed Computing Systems*, pp. 104-111, 1988.

[11] K. Kennedy, Compilers, Languages, and Libraries, in The Grid: Blueprint for a New Computing Infrastructure, ed. by I. Foster and C. Kesselman, pp. 181-204, Morgan Kaufmann, 1998.

[12] M. Corbin and P. Sapaty, "Distributed object-based simulation in WAVE," *Simulation Practice and Theory*, vol. 3, no. 3, pp. 157-181, Nov. 1995.

[13] R. S. Gray, "Agent Tcl: a flexible and secure mobile-agent system," in *Proc. of the 4th Annual Tcl/Tk Workshop, TCL 96,* pp. 374-377, Monterey, California, Jul. 1996.

[14] D. Johansen, R. van Renesse, and F. B. Schneider, *An Introduction to the TACOMA Distributed System Version 1.0*, Technical Report 05-23, Department of Computer Science, University of Troms, Jun. 1995.

[15] D. B. Lange and D. T. Chang, *IBM Aglets Workbench, Programming Mobile Agents in Java*, White Paper, IBM Tokyo Research, Japan, Sep. 1996.

[16] A. Villazón, *A Reflective Architecture Applied to Mobile Code Environments*, Ph.D. thesis, University of Geneva, Dec. 2001.

[17] M. Fukuda, L. F. Bic, and M. B. Dillencourt, "Messages versus messengers in distributed programming," *J. of Distributed Programming*, vol. 57, no. 2, pp. 188-211, Nov. 1999.

[18] M. Fukuda, L. F. Bic, M. B. Dillencourt, and F. Merchant, "Distributed coordination with messengers," *Science of Computer Programming*, vol. 31, no. 2-3, pp. 291-311, Jul. 1998.

[19] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, *Alchemi: A .NET-Based Grid Computing Framework and its Integration into Global Grids*, Technical Report, GRIDS-TR-2003-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, Dec. 2003.

[20] G. Cabri, L. Leonardi, and F. Zambonelli, *Weak and Strong Mobility in Mobile Agent Applications*, Dipartimento di Scienze dell'Ingegneria , Università di Modena e Reggio Emilia Via Campi, Modena, Italy, 2000.

[21] Northeast Parallel Architectures Center and RSA Factoring-By-Web project.

[22] I. Foster, J. Geisler, B. Nickless, W. Smith, and S.Tuecke, "Software infrastructure for the I-WAY high performance distributed computing experiment," in *Proc. 5th IEEE Symp. on High Performance Distributed Computing*, pp. 562-571, Aug. 1997.

[23] Globus project website, http://www.globus.org.

[24] M. Fukuda, Y. Tanaka, N. Suzuki, L. F. Bic, S. Kobayashi, "A mobile-agent-based PC grid," in *Autonomic Computing Workshop*, pp. 142-150, Jun 2003.

[25] L. Chunlin and L. Layuan, "An agent-based approach for grid computing," Proc. of the Fourth Int. Conf. on Parallel and Distributed Computing, Applications and Technologies, pp. 608-611, Aug. 2003.

[26] T. Y. Li, Z. G. Zhao, and S. Z. Yo, "A-peer: an agent platform integrating peer-to-peer network," in *Proc. 3rd IEEE Int. Symp. on Cluster Computing and the Grid, CCGrid*, pp. 614-617, 2003.

[27] D. Bruneo, M. Scarpa, A. Zaia, and A. Puliafito, "Communication paradigms for mobile grid users," in *Proc. 3rd IEEE Int. Symp. on Cluster Computing and the Grid, CCGrid*, pp. 669-677, 2003.

[28] L. Moreau, *et al.*, "On the use of agents in bioinformatics grid," in *Proc. 3rd IEEE Int. Symp. on Cluster Computing and the Grid, CCGrid*, pp. 653-668, May 2003.

[29] J. Cao, *et al.*, "Agent-based resource management for grid computing," in *Proc. 2nd IEEE Int. Symp. on Cluster Computing and the Grid, CCGrid*, pp. 350-351, May 2002.

[30] S. A. Elfayoumy and J. H. Graham, "An agent-based architecture for tuning parallel and distributed applications performance," in *Proc. of the 2nd Workshop on Cluster-Based Computing*, May 2000.

[31] L. Serafini, H. Stockinger, K. Stockinger, and F. Zini, "Agent-based query optimisation in a grid environment," in *Proc. IASTED Int. Conf. on Applied Informatics*, Innsbruck, Austria, Feb. 2001.

[32] B. J. Overeinder, F. M. T. Brazier, and O. Marin, "Fault tolerance in scalable agent support systems integrating DARX in the agentscape framework," in *Proc. 3rd IEEE Int. Symp. on Cluster Computing and the Grid, CCGrid*, pp. 688-696, May 2003.

[33] M. L. Aird, W. B. Medina, and J. Padget," MONET: service discovery and composition for mathematical problems," in *Proc. 3rd IEEE In. Symp. on Cluster Computing and the Grid, CCGrid*, pp. 678-684, May 2003.

[34] S. Corsava and V. Getov, "Agent-based service management in large datacentres and grids," in *3rd IEEE International*

1. Speedup
2. Fast Fourier Transform
3. Binary Space Partitioning Tree
4. Constructive Solid Geometry

Symposium on Cluster Computing and the Grid, CCGrid, pp. 633-640, May 2003.
[35] C. Wicke, L. F. Bic, M. B. Dillencourt, and M. Fukuda, "Automatic state capture of self-migrating computations in messengers," in *Proc. 2nd International Conf. on Mobile Agents, MA '98, Lecture Notes in Computer Science*, vol. 1477, pp. 68-79, *Springer-Verlag, Berlin, Germany*, Sep. 1998.
[36] E. Gendelman, *Messengers User's Manual (Version 2.1)*, Information & Computer Science, University of California, Irvine, California, 2001.
[37] L. F. Bic, M. Fukuda, and M. B. Dillencourt, "Distributed computing using autonomous objects," *IEEE Computer*, vol. 29, no. 8, pp. 55-61, Aug. 1996.
[38] M. Gmelin, J. Kreuzinger, M. Pfeffer, and Th. Ungerer, "Agent-based distributed computing with Jmessengers," *I2CS Innovative Internet Computing Systems*, pp. 131-145, Ilmenau, Springer-Verlag LNCS 2060, Feb. 2001.
[39] Alchemi.NET Site, http://www.alchemi.net, 2004.
[40] S. Rahimi, A. Narayanan, and M. Sabharwal, "MPIAB: a novel agent architecture for parallel processing," in *Proc. IEEE/WIC Int. Conf. on Intelligent Agent Technology, IAT'03*, p. 554, Oct. 2003.

روح‌الله مافی شرح حال علمی مولف در زمان انتشار نشریه در دسترس نبود.

حسین دلداری کارشناسی خود را در رشته فیزیک از دانشگاه فردوسی مشهد در سال ۱۳۵۱ اخذ نمود و سپس کارشناسی ارشد را در سال ۱۳۵۷ در رشته علوم کامپیوتر از دانشگاه ایالتی اورگان در شهر یوجین در کشور امریکا اخذ نمود. دکترای ایشان در رشته پردازش موازی از دانشگاه لیدز انگلستان می‌باشد که در سال ۱۳۷۵ موفق به دریافت آن گردید. از سال ۱۳۵۸ به بعد به تدریس در رشته مهندسی کامپیوتر دانشگاه فردوسی مشهد مشغول بوده است که تاکنون ادامه دارد. زمینه تحقیقاتی نامبرده الگوریتم‌های موازی، سیستم‌های توزیع‌شده، محاسبات گریدی و کامپایلرها می‌باشد. حاصل کار تحقیقاتی ایشان ارائه چندین مقاله در کنفرانس‌های داخلی و خارجی و نیز چاپ چند مقاله در مجلات علمی-پژوهشی داخل و خارج کشور می‌باشد.