پیاده‌سازی روش گرادیان مزدوج با کارایی بالا به کمک زبان آزاد محاسباتی روی پردازنده‌های گرافیکی

فرشید مسیبی *

گروه مهندسی عمران، دانشکده فنی و مهندسی، دانشگاه اصفهان

(دریافت مقاله: ۱۳۹۱/۰۹/۱٤- دریافت نسخه نهایی: ۱۳۹۲/۸/۱۲)

چکیده - بیشتر روش‌های عددی حل معادلات دیفرانسیل پاره‌ای مجهولات مورد نیاز خود را ازطریق حل یک دستگاه معادلات خطی به‌دست می‌آورند. این بخش از روند حل در این روش‌ها قسمت قابل توجهی از زمان حل را به خود اختصاص می‌دهد. از این رو حل سریع‌تر دستگاه‌های معادلات خطی همواره مورد توجه محققان بوده و تحقیقات بسیاری در این زمینه انجام شده و یا در حال انجام است. در این تحقیق با کمک یک هسته محاسباتی قابل تنظیم پیاده‌سازی شده توسط زبان آزاد محاسباتی، دستگاه‌های معادلات خطی با استفاده از پردازنده‌های گرافیکی حل می‌شود. برای حل این دستگاه‌ها از روش گرادیان مزدوج استفاده شده است. پارامترهای هسته‌های محاسباتی به نحوی تنظیم می‌شوند که حل دستگاه به سریع‌ترین شکل ممکن انجام یابد. برای کارایی هر چه بیشتر در این روش، دو شکل از روش گرادیان مزدوج که بیشترین همخوانی را با مدل اجرا در زبان آزاد محاسباتی دارد، پیاده‌سازی شده و عملکرد آنها با کتابخانه توابع وینا سی ال روی پردازنده و پردازنده گرافیکی مقایسه می‌شود. در هر دو شکل روش هسته‌های محاسباتی تا حد امکان با یکدیگر ترکیب شده تا از زمان اضافی صرف شده برای فراخوانی هسته‌ها کاسته شود. نتایج نشان می‌دهد روش پیشنهادی روی تمام سیستم‌ها و مسائل مورد بررسی بسیار بهتر از کتابخانه توابع وینا سی ال عمل می‌نماید.

واژگان کلیدی: حل دستگاه‌های معادلات خطی، روش گرادیان مزدوج، پردازنده‌های گرافیکی، زبان آزاد محاسباتی.

High Performance Implementation of Conjugate Gradient Method Using OpenCL on Graphics Processing Units

F. Mossaiby

Department of Civil Engineering, Faculty of Engineering, University of Isfahan

Abstract: ***Most solution methods for Partial Differential Equations (PDEs) find their unknowns through solving a linear system of equations. This step consumes a considerable part of total solution time, and hence accelerating the solution of linear systems of equations has been the subject of many researches. In this research we solve linear systems of equations on Graphics***

* : مسئول مکاتبات، پست الکترونیکی: mossaiby@eng.ui.ac.ir

Processing Units (GPUs) using a tunable Sparse Matrix-Vector multiplication (SpMV) implemented in Open Computing Language (OpenCL). We use the Conjugate Gradient (CG) method to this end. Kernel parameters are set such that the linear system of equation is solved in the fastest way possible. For a better performance, two variants of CG which most comply the execution model in OpenCL are implemented and their performances are compared with ViennaCL library on CPU and GPU. In both variants, the kernels are fused to reduce the kernel launch overhead. The results show that the proposed method consistently outperforms the ViennaCL library on a wide range of test systems and problems.

Keywords: ***Solution of linear systems of equations, Conjugate Gradient method, Graphics processing unit (GPU), Open computing language (OpenCL).***

۱- مقدمه

حل دستگاه‌های معادلات خطی قسمت عمده‌ای از روند حل را در بیشتر روش‌های عددی تشکیل می‌دهد. امروزه با پیشرفت علوم مهندسی و به‌تبع آن پیچیدگی روز افزون مسائل، حل مسائلی با میلیون‌ها درجه آزادی بسیار عادی به‌نظر می‌رسد. با افزایش درجات آزادی، زمان حل نیز به‌شدت افزایش می‌یابد، از اینرو محققان همواره تلاش کرده‌اند حل دستگاه معادلات خطی را به سریع‌ترین شکل ممکن انجام دهند. حاصل این تلاش‌ها به‌وجود آمدن تعداد زیادی روش برای حل این دستگاه‌ها است [۱ و ۲]. این روش‌ها به دو دسته عمده روش‌های مستقیم[۱] و روش‌های تکراری[۲] تقسیم می‌شوند. به‌علت پیچیدگی بالا و نیاز به حافظه زیاد در روش‌های مستقیم، در مسائل بزرگ معمولاً از روش‌های تکراری استفاده می‌شود. از مهم‌ترین روش‌های تکراری می‌توان به‌روش گرادیان مزدوج[۳] و یا گرادیان مزدوج دوگانه[۴] اشاره کرد که همراه با سایر روش‌هایی از این دست، در زمره روش‌های زیرفضای کرایلف[۵] جای می‌گیرند [۱ و ۲]. روش گرادیان مزدوج به‌علت سادگی پیاده‌سازی بیشترین کاربرد را در بین روش‌های تکراری دارا است. در این روش ماتریس ضرائب باید متقارن و مثبت معین[۶] باشد. از آنجا که ماتریس ضرائب در برخی از روش‌های عددی فاقد این شرایط است، به‌جای این روش از روش گرادیان مزدوج دوگانه استفاده می‌شود. این روش به لحاظ عددی ممکن است دچار مشکلاتی از جمله ناپایداری عددی گردد که با استفاده از روش گرادیان مزدوج دوگانه پایدار شده[۷] این مشکل نیز برطرف می‌گردد. روش گرادیان مزدوج دوگانه از لحاظ ساختار و پیاده‌سازی شباهت بسیار زیادی به‌روش گرادیان مزدوج دارد.

علاوه بر تحقیقاتی که در جهت توسعه الگوریتم‌های حل دستگاه‌های معادلات خطی انجام شده است، محققان همواره سعی کرده‌اند از نظر سخت‌افزاری نیز سرعت محاسبات عددی را افزایش دهند. این روش‌ها را که عمدتاً به‌عنوان محاسبات با کارایی بالا[۸] از آنها یاد می‌شود می‌توان به سه دسته عمده تقسیم نمود. در دسته اول از تعدادی رایانه مستقل که توسط ابزاری مانند شبکه‌های محلی[۹] به‌یکدیگر متصل شده‌اند برای شکستن روند حل مسئله و تقسیم آن بین رایانه‌ها استفاده می‌شود. به‌جهت مستقل بودن حافظه رایانه‌های مورد استفاده در این روش، اصطلاحاً به آن حافظه گسترده[۱۰] گفته می‌شود. قابلیت توسعه سامانه‌ها یکی از مهم‌ترین مزایای این دسته است. با توسعه پردازنده‌های چند هسته‌ای[۱۱] در سال‌های اخیر، نوع دیگری از محاسبات با کارایی بالا در میان محققان رواج پیدا کرده است. در این روش‌ها بخشی از مسئله (همانند حلقه‌های موجود در روند برنامه) میان هسته‌های یک پردازنده تقسیم شده و در نتیجه روند حل مسئله تسریع می‌گردد. برخلاف دسته قبل حافظه در دسترس برای تمامی هسته‌های پردازنده مشترک بوده و از اینرو این دسته از روش‌ها، حافظه مشترک[۱۲] نامیده می‌شود. دسته سوم که اخیراً برای انجام محاسبات با سرعت بالا مورد توجه قرار گرفته است، استفاده از کمک‌پردازنده‌های محاسباتی[۱۳] است. در این روش‌ها از یک ابزار کمکی مانند پردازنده‌های گرافیکی[۱۴] برای انجام بخشی از محاسبات استفاده می‌شود.

در سال‌های اخیر با رشد بسیار چشمگیر قدرت محاسبات پردازنده‌های گرافیکی و قیمت متعادل آنها، استفاده از آنها در انجام محاسبات عددی بسیار رایج شده است. پردازنده‌های

گرافیکی فعلی دارای صدها هسته هستند که به آنها قدرت محاسباتی دهها برابر پیشرفته‌ترین پردازنده‌های موجود در بازار می‌بخشد. پیش از رایج شدن استفاده از پردازنده‌های گرافیکی برای محاسبات عددی این پردازنده‌ها تنها قادر به انجام اعمال گرافیکی بودند، بنابراین محققان با شبیه‌سازی برخی عملیات مورد نیاز با اعمال گرافیکی از کتابخانه‌های توابع گرافیکی بدین منظور استفاده می‌کردند [۳]. واضح است که انجام این کار بسیار پیچیده بود و تنها امکان انجام برخی اعمال محدود توسط پردازنده‌های گرافیکی وجود داشت. با رشد سریع قدرت پردازنده‌های گرافیکی و رایج‌تر شدن استفاده از آنها در انجام محاسبات، شرکت‌های سازنده این پردازنده‌ها شروع به گسترش قابلیت‌های آنها نموده و قابلیت پردازش عمومی روی پردازنده‌های گرافیکی[۱۵] را فراهم نمودند. از موفق‌ترین این بسترهای نرم‌افزاری که امروزه کاربرد بسیار زیادی در بین محققین جهت انجام محاسبات عددی پیدا کرده است، کودا[۱۶] [۴] متعلق به شرکت انویدیا[۱۷] است. کودا پس از ارائه‌اش در اوائل سال ۲۰۰۷ پیشرفت چشمگیری داشته و امکانات و قابلیت‌های آن به‌شدت در حال افزایش بوده است. مشکلی که با ارائه این بسترها به‌وجود آمد، وابستگی کامل آنها به سخت‌افزارهای شرکت سازنده بود که امکان استفاده از یک برنامه نوشته شده توسط یکی از آنها روی دیگر سخت‌افزارها را غیرممکن می‌نمود. برای حل این مشکل در اواسط سال ۲۰۰۸ شرکت اپل[۱۸] با همراهی شرکت‌های بزرگ تولید کننده پردازنده‌های گرافیکی اقدام به معرفی زبان آزاد محاسباتی[۱۹] [۵] نمود. این زبان یک استاندارد باز[۲۰] است که با استفاده از آن برنامه‌نویسان می‌توانند با استفاده از یک زبان و کتابخانه نرم‌افزاری واحد برنامه‌هایی تولید کنند که قابل اجرا روی تمام سخت‌افزارهای سازگار باشد. ارائه زبان آزاد محاسباتی تحولی بزرگ در زمینه بسترهای نرم‌افزاری مرتبط با پردازنده‌های گرافیکی بود، به‌نحوی که برخی از شرکت‌ها تصمیم به‌عدم توسعه بسترهای نرم‌افزاری خاص سخت‌افزارهای خود گرفتند و در عوض به توسعه هرچه بیشتر زبان آزاد محاسباتی پرداختند. به‌علت ساختار بسیار پیچیده پردازنده‌های گرافیکی واضح است که برنامه‌نویسی برای آنها با مدل‌های عادی برنامه‌نویسی برای پردازنده‌ها بسیار متفاوت است. این تفاوت را به‌خوبی می‌توان در مدل برنامه‌نویسی کودا و یا زبان آزاد محاسباتی مشاهده نمود. برای آشنایی بیشتر با نحوه برنامه‌نویسی پردازنده‌های گرافیکی خواننده می‌تواند به مراجع [۶-۸] مراجعه نماید.

۲- روش گرادیان مزدوج و تحقیقات مرتبط

گاوس در سال ۱۸۲۳ اولین روش تکراری برای حل دستگاه‌های معادلات خطی را ابداع نمود [۲]. روش‌های تکراری در طول زمان گسترش بسیار زیادی یافته‌اند. روش گرادیان مزدوج یکی از پرکاربردترین این روش‌ها است که در سال ۱۹۵۲ توسط هستنس و استایفل ابداع شد [۹]. توضیحات بیشتر در مورد روش گرادیان مزدوج خارج از حوصله این نوشتار است و خواننده می‌تواند برای اطلاعات بیشتر به مرجع [۱۰] مراجعه نماید.

تلاش‌های بسیار زیادی برای افزایش سرعت این روش روی پردازنده‌ها و پردازنده‌های گرافیکی انجام شده است که از آن جمله می‌توان به مراجع [۱۱-۱۴] اشاره نمود. در تمامی این مراجع از زبان برنامه‌نویسی کودا استفاده شده است که آنها را به سخت‌افزارهای شرکت انویدیا وابسته می‌سازد. تا جایی که نگارنده اطلاع دارد، تنها کتابخانه توابعی که بر مبنای زبان آزاد محاسباتی برای حل دستگاه‌های معادلات خطی به‌کار می‌رود، کتابخانه توابع وینا سی ال است [۱۵]. این کتابخانه رایگان / متن باز مجموعه بسیار کاملی از توابع برای حل دستگاه‌های معادلات خطی تنک به‌همراه اعمال ابتدایی برداری و ماتریسی ارائه می‌نماید که در نوع خود بی‌نظیر است.

باید اشاره نمود که شکل‌های بسیار متفاوتی از روش گرادیان مزدوج ارائه شده است که با وجود هم‌ارز بودن از نظر همگرایی و جواب نهایی، هر یک خصوصیات ویژه خود را دارا می‌باشند. از جمله این خصوصیت‌ها می‌توان به امکان تجمیع

برخی محاسبات و یا تعداد دفعات دسترسی به یک بردار در طی هر تکرار الگوریتم اشاره نمود. این خصوصیات باعث می‌شود هر یک از این شکل‌ها برای یک سخت‌افزار خاص مناسب‌تر باشند. تحقیقات زیادی در این زمینه انجام شده و یا در حال انجام است [۱۶].

شکل ۱ الگوریتم اصلی روش گرادیان مزدوج را به‌همراه دو شکل پیاده‌سازی شده آن در این تحقیق (الگوریتم ۶.۱۸ از مرجع [۱] و الگوریتم ۲.۳ از مرجع [۱۷]) نشان می‌دهد. در تمامی این شکل‌ها برای حل دستگاه معادلات خطی $Ax = b$، ابتدا یک فرض اولیه x_0 درنظر گرفته می‌شود و پس از چند محاسبه اولیه ساده، روند کلی تا همگرایی لازم ادامه می‌یابد. در غیاب اطلاعات درباره دستگاه معادلات خطی موردنظر، $x = 0$ فرض می‌شود که باعث ساده‌سازی محاسبات اولیه می‌گردد.

۳- پیاده‌سازی ضرب ماتریس در بردار

همان‌گونه که در شکل ۱ مشاهده می‌شود، مهم‌ترین عملیات مورد نیاز در همه شکل‌های پیاده‌سازی شده از روش گرادیان مزدوج، ضرب ماتریس در بردار و نیز ضرب داخلی دو بردار است که آنها نیز باید در زبان آزاد محاسباتی پیاده‌سازی شوند. مهم‌ترین جزء تشکیل دهنده تمامی روش‌های زیرفضای کرایلف از جمله روش گرادیان مزدوج که بیش از ۸۰٪ از زمان حل را به‌خود اختصاص می‌دهد، هسته محاسباتی[۲۱] ضرب ماتریس در بردار است. از آنجا که بیشتر ماتریس‌های حاصل از روش‌های عددی تنک[۲۲] می‌باشد، بیشتر تلاش‌ها در این زمینه به حل این گروه از دستگاه‌های معادلات خطی اختصاص دارد. در این تحقیق با استفاده از یک هسته محاسباتی ضرب ماتریس‌های تنک در بردار با کارایی بالای پیشنهاد شده توسط نگارنده [۶]، اقدام به پیاده‌سازی دو شکل از روش گرادیان مزدوج برای دستگاه‌های معادلات خطی شده است. پیاده‌سازی این هسته به‌گونه‌ای است که با انتخاب بهینهٔ پارامترهای آن، کارایی الگوریتم را افزایش داد. توضیحات بیشتر در این زمینه در قسمت بعد خواهد آمد.

۴- پیاده‌سازی ضرب داخلی بردارها

پیاده‌سازی ضرب داخلی دو بردار با کارایی بالا در زبان آزاد محاسباتی (برخلاف تصور اولیه) چندان ساده نیست. عملیات ضرب داخلی دو بردار از دو قسمت مجزای محاسبه حاصل‌ضرب درایه‌های متناظر دو بردار و سپس محاسبه حاصل‌جمع تمام این حاصل‌ضرب‌های میانی تشکیل شده است. به‌علت مدل برنامه‌نویسی خاص زبان آزاد محاسباتی، قسمت اول به‌سادگی انجام می‌شود، اما قسمت دوم (که اصطلاحاً عملیات کاهش[۲۳] نامیده می‌شود) به‌سادگی قابل انجام نیست. برای محاسبه حاصل‌جمع در مدل برنامه‌نویسی زبان آزاد محاسباتی، از روش کاهش موازی[۲۴] استفاده می‌شود. شکل ۲ نحوه انجام این کار را نشان می‌دهد. در این روش در هر گروه کاری[۲۵] در هر مرحله هر یک از درایه‌های نیمه اول با درایه متناظر در نیمه دوم جمع شده و در درایه مربوط به نیمه اول قرار می‌گیرد. با این کار تعداد درایه‌ها به نصف کاهش می‌یابد. در مرحله دوم این نیمه همانند یک گروه کاری درنظر گرفته می‌شود و طی یک مرحله دیگر تعداد درایه‌ها باز هم به نصف کاهش می‌یابد. این کار تا زمانی ادامه می‌یابد که حاصل‌جمع تمام درایه‌ها در درایه اول جمع گردد. این حاصل جمع در محل مربوط در بردار خروجی ذخیره می‌گردد. با انجام این مراحل به‌صورت کامل، تعداد درایه‌ها از mn به n کاهش می‌یابد که در آن m تعداد رشته‌های اجرا در هر گروه کاری و n تعداد گروههای کاری است. چنانچه لازم باشد تمامی عملیات روی پردازنده گرافیکی انجام شود، این کار تا زمانی انجام می‌شود که تنها یک درایه باقی بماند. اما از آنجا که در تحقیق حاضر حاصل‌جمع نهایی جهت استفاده روی پردازنده مورد نیاز است و نیز با توجه به زمان اضافی صرف شده جهت فراخوانی هسته محاسباتی مربوطه، تصمیم بر این شد که یک مرحله عملیات کاهش روی پردازندهٔ گرافیکی

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0, \quad \mathbf{p}_0 = \mathbf{r}_0$$
for $j = 0$ until convergence
$$\gamma_j = \frac{(\mathbf{r}_j, \mathbf{r}_j)}{(\mathbf{A}\mathbf{p}_j, \mathbf{p}_j)}$$
$$\mathbf{x}_{j+1} = \mathbf{x}_j + \gamma_j \mathbf{p}_j$$
$$\mathbf{r}_{j+1} = \mathbf{r}_j - \gamma_j \mathbf{A}\mathbf{p}_j$$
$$\rho_j = \frac{(\mathbf{r}_{j+1}, \mathbf{r}_{j+1})}{(\mathbf{r}_j, \mathbf{r}_j)}$$
$$\mathbf{p}_{j+1} = \mathbf{r}_{j+1} + \rho_j \mathbf{p}_j$$
end

الف- شکل اصلی روش گرادیان مزدوج (الگوریتم ۶.۱۷ از مرجع [۱])

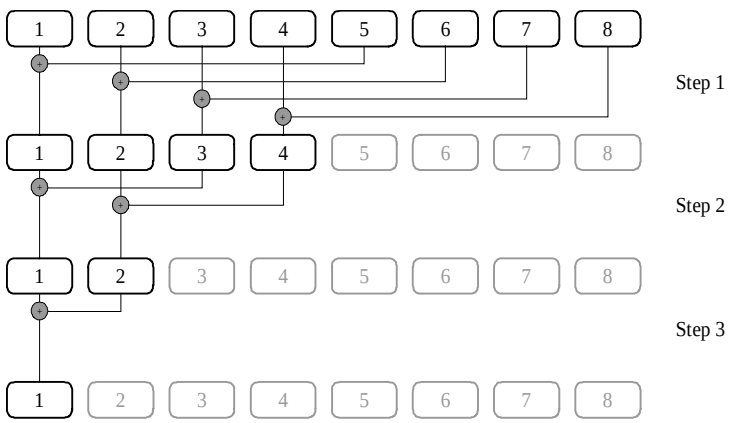
$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0, \quad \mathbf{x}_{-1} = \mathbf{0}, \quad \rho_0 = 1$$
for $j = 0$ until convergence
$$\gamma_j = \frac{(\mathbf{r}_j, \mathbf{r}_j)}{(\mathbf{A}\mathbf{r}_j, \mathbf{r}_j)}$$
$$\text{if } (j > 0),\ \rho_j = \left(1 - \frac{\gamma_j}{\gamma_{j-1}} \frac{(\mathbf{r}_j, \mathbf{r}_j)}{(\mathbf{r}_{j-1}, \mathbf{r}_{j-1})} \frac{1}{\rho_{j-1}}\right)^{-1}$$
$$\mathbf{x}_{j+1} = \rho_j(\mathbf{x}_j - \gamma_j \mathbf{r}_j) + (1 - \rho_j)\mathbf{x}_{j-1}$$
$$\mathbf{r}_{j+1} = \rho_j(\mathbf{r}_j - \gamma_j \mathbf{A}\mathbf{r}_j) + (1 - \rho_j)\mathbf{r}_{j-1}$$
end

ب- شکل اول روش گرادیان مزدوج (الگوریتم ۶.۱۸ از مرجع [۱])

$$\mathbf{p}_0 = \mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0, \quad \gamma_0 = \frac{(\mathbf{r}_0, \mathbf{r}_0)}{(\mathbf{A}\mathbf{r}_0, \mathbf{r}_0)}, \quad \rho_{-1} = 0$$
for $j = 0$ until convergence
$$\mathbf{p}_j = \mathbf{r}_j + \rho_{j-1}\mathbf{p}_{j-1}$$
$$\mathbf{q}_j = \mathbf{A}\mathbf{r}_j + \rho_{j-1}\mathbf{q}_{j-1}$$
$$\mathbf{x}_{j+1} = \mathbf{x}_j + \gamma_j \mathbf{p}_j$$
$$\mathbf{r}_{j+1} = \mathbf{r}_j - \gamma_j \mathbf{q}_j$$
$$\rho_j = \frac{(\mathbf{r}_{j+1}, \mathbf{r}_{j+1})}{(\mathbf{r}_j, \mathbf{r}_j)}$$
$$\gamma_j = \frac{(\mathbf{r}_{j+1}, \mathbf{r}_{j+1})}{(\mathbf{A}\mathbf{r}_{j+1}, \mathbf{r}_{j+1}) - \frac{\rho_j}{\gamma_j}(\mathbf{r}_{j+1}, \mathbf{r}_{j+1})}$$
end

ج- شکل دوم روش گرادیان مزدوج (الگوریتم ۲.۳ از مرجع [۱۷])

شکل ۱- شکل‌های پیاده‌سازی شده متفاوت از الگوریتم گرادیان مزوج



شکل ۲- عملیات کاهش موازی

انجام شده و حاصل‌جمع نهایی روی پردازنده (با استفاده از استاندارد چندپردازنده باز[۲۶] برای افزایش سرعت) محاسبه شود. هم‌چنین برای افزایش کارایی ابتدا بخشی از عمل جمع به‌صورت عادی (بدون استفاده از کاهش موازی) انجام شده و سپس نتایج حاصله در کاهش موازی مورد استفاده قرار می‌گیرد. شکل ۳ متن هسته محاسباتی برای انجام ضرب داخلی دو بردار را نشان می‌دهد. لازم به‌ذکر است که کارایی بهینه عملیات ضرب دو ماتریس با اندازه خاصی برای هر گروه کاری به‌دست می‌آید که این اندازه به عوامل بسیاری از جمله اندازه بردار موردنظر بستگی دارد. در تحقیق حاضر اندازه هر گروه کاری با استفاده از ماکروهای پیش‌پردازنده[۲۷] به‌صورت یک پارامتر درنظر گرفته شده است. در زمان اجرا با تغییر این پارامتر در یک محدوده و زمان‌گیری دقیق، مقدار بهینه آن در شرایط کاملاً واقعی به‌دست می‌آید. از آنجا که برای حل یک دستگاه معادلات خطی نیاز به صدها و یا هزاران بار استفاده از ضرب داخلی بردارها است، چند بار استفاده از ضرب داخلی برای پیدا کردن مقدار بهینه پارامترها کاملاً منطقی است. هم‌چنین در صورتی‌که مقدار بهینه قبلاً به‌طریقی (مثلاً از اجرای قبلی) به‌دست آمده باشد، دیگر نیازی به انجام این مرحله نیست. در بسیاری از روش‌های عددی (مانند روش‌های پیش‌روی زمانی[۲۸]) نیاز به حل یک دستگاه معادلات خطی در هر گام زمانی وجود دارد. در بیشتر موارد با وجود تغییر دستگاه معادلات خطی در گام زمانی، ابعاد دستگاه معادلات خطی و توزیع عناصر غیرصفر در آن ثابت باقی می‌ماند. از آنجا که پارامترهای هسته‌های ضرب ماتریس تنک در بردار و ضرب داخلی دو بردار تنها به‌همین عوامل وابسته‌اند، با یک بار بهینه‌سازی آنها در ابتدا، حل تمامی دستگاه‌ها به‌نحو بهینه انجام خواهد گرفت که بیش از پیش انجام این مرحله بهینه‌سازی را مقرون به‌صرفه می‌سازد.

۵- بهینه‌سازی الگوریتم گرادیان مزدوج برای استفاده در زبان آزاد محاسباتی

معیار انتخاب شکل مناسب الگوریتم گرادیان مزدوج تعداد، نوع و محل عملیات به‌کار رفته در آن روش و در نتیجه امکان پیاده‌سازی بهینه آن با توجه به مدل برنامه‌نویسی و اجرای زبان آزاد محاسباتی است. به‌عنوان مثال در شکل ۱- الف شکل اصلی روش گرادیان مزدوج نشان داده شده است. در این شکل سه عمل بروزرسانی بردار[۲۹] وجود دارد که توسط یک عمل ضرب داخلی به دو قسمت تقسیم شده است. چنانچه بتوان اعمال بروزرسانی بردارها را به‌صورت پشت سر هم انجام داد، می‌توان آنها را در یک هسته محاسباتی متمرکز نمود و از صرف زمان اضافه برای فراخوانی هسته محاسباتی اجتناب نمود. دو شکل دیگر الگوریتم گرادیان مزدوج نشان داده شده دارای چنین خواصی هستند. در هر دوی این شکل‌ها تمامی اعمال بروزرسانی بردارها پشت سر هم قرار دارد و بنابراین می‌تواند در یک هسته محاسباتی متمرکز شود. این کار در اصطلاح آمیختن هسته‌ها[۳۰] نامیده می‌شود. مورد دیگری که در اینجا قابل ذکر است، عمل ضرب داخلی بردارها می‌باشد. در هر سه شکل ارائه شده از روش گرادیان مزدوج دو عمل ضرب داخلی وجود دارد. اما در شکل اصلی و شکل دوم، بردارهای مورد نیاز در عمل ضرب داخلی کاملاً از یکدیگر مجزا هستند، اما در شکل اول یکی از بردارهای مورد استفاده در دو عمل ضرب داخلی با یکدیگر مشترک است. علاوه بر این، دو عمل ضرب داخلی در کنار یکدیگر مورد استفاده قرار می‌گیرند که مجموع این دو نکته را می‌توان جهت آمیختن دو هسته ضرب داخلی و علاوه بر آن کاهش دسترسی به حافظه اصلی پردازنده گرافیکی مورد استفاده قرار داد. از آنجا که سرعت دسترسی به حافظه اصلی در پردازنده‌های گرافیکی نسبت به‌سایر انواع حافظه در این پردازنده‌ها بسیار کمتر است [۸-۶]، این کار می‌تواند در افزایش سرعت محاسبات نقش مؤثری را ایفا کند.

۶- مثال‌های عددی

جدول ۱ مشخصات سیستم‌های رایانه‌ای مورد استفاده در این تحقیق را نشان می‌دهد. همان‌گونه که پیش‌تر گفته شد،

```
__kernel void
__attribute__((reqd_work_group_size(WORKGROUP_SIZE, 1, 1)))
InnerProd(
__global double const *X_Values,
__global double const *Y_Values,
__global double *Z_Values,
int N,
__local double *Buffer
)
{
   int gid = get_global_id(0);

   // Serial part
   double Accumulator = 0.00;

   while (gid < N)
   {
      Accumulator += X_Values[gid] * Y_Values[gid];
      gid += get_global_size(0);
   }

   // Parallel part
   int lid = get_local_id(0);
   Buffer[lid] = Accumulator;

   barrier(CLK_LOCAL_MEM_FENCE);

#if WORKGROUP_SIZE > 512

   if (lid < 512)
   {
      Buffer[lid] += Buffer[lid + 512];
   }

   barrier(CLK_LOCAL_MEM_FENCE);

#endif

#if WORKGROUP_SIZE > 256

   if (lid < 256)
   {
      Buffer[lid] += Buffer[lid + 256];
   }

   barrier(CLK_LOCAL_MEM_FENCE);

#endif

#if WORKGROUP_SIZE > 128

   if (lid < 128)
   {
      Buffer[lid] += Buffer[lid + 128];
   }

   barrier(CLK_LOCAL_MEM_FENCE);

#endif
```

```
#if WORKGROUP_SIZE > 64

   if (lid < 64)
   {
      Buffer[lid] += Buffer[lid + 64];
   }

   barrier(CLK_LOCAL_MEM_FENCE);

#endif

#if WORKGROUP_SIZE > 32

   if (lid < 32)
   {
      Buffer[lid] += Buffer[lid + 32];
   }

   barrier(CLK_LOCAL_MEM_FENCE);

#endif

#if WORKGROUP_SIZE > 16

   if (lid < 16)
   {
      Buffer[lid] += Buffer[lid + 16];
   }

   barrier(CLK_LOCAL_MEM_FENCE);

#endif

#if WORKGROUP_SIZE > 8

   if (lid < 8)
   {
      Buffer[lid] += Buffer[lid + 8];
   }

   barrier(CLK_LOCAL_MEM_FENCE);

#endif

#if WORKGROUP_SIZE > 4

   if (lid < 4)
   {
      Buffer[lid] += Buffer[lid + 4];
   }

   barrier(CLK_LOCAL_MEM_FENCE);

#endif

#if WORKGROUP_SIZE > 2

   if (lid < 2)
```

```
   {
      Buffer[lid] += Buffer[lid + 2];
   }

   barrier(CLK_LOCAL_MEM_FENCE);

#endif

#if WORKGROUP_SIZE > 1

   if (lid < 1)
   {
      Buffer[lid] += Buffer[lid + 1];
   }

   barrier(CLK_LOCAL_MEM_FENCE);

#endif

   // Store final result
   if (lid == 0)
   {
      Z_Values[get_group_id(0)] = Buffer[0];
   }
}
```

شکل ۳- هسته محاسباتی ضرب داخلی بردارها

جدول ۱- مشخصات سیستم‌های رایانه‌ای مورد استفاده

ردیف	پردازنده		حافظه	سیستم عامل	پردازنده گرافیکی		بستر نرم‌افزاری زبان آزاد محاسباتی
	نوع	تعداد هسته‌ها	اصلی		نوع	حافظه اصلی	
۱	AMD Phenom Quad core 9950	۴	4 GB DDR2	Ubuntu 10.10 (64 bit Linux)	NVIDIA GeForce GTX 280	1 GB GDDR5	NVIDIA CUDA 5.0
۲	Intel Core2 Quad Q8300	۴	4 GB DDR2	Ubuntu 12.10 (64 bit Linux)	NVIDIA GeForce GTX 550 Ti	1.5 GB GDDR3	NVIDIA CUDA 5.0
۳	Intel Core i7 2700	۴ + ۴	8 GB DDR3	Ubuntu 12.10 (64 bit Linux)	AMD Radeon HD 6970	2 GB GDDR5	AMD APP SDK 2.7

بیش از ۸۰٪ زمان حل دستگاه‌های معادلات خطی در روش‌های تکراری صرف ضرب ماتریس تنک در بردار می‌شود که سرعت دسترسی به حافظه مهم‌ترین نقش را در کارایی آن ایفا می‌کند. برای داشتن برآوردی از سرعت حافظه در پردازنده‌ها از آزمون استریم[۳۱] [۱۸] و در پردازنده‌های گرافیکی از آزمونی مشابه که توسط نگارنده با زبان آزاد محاسباتی پیاده‌سازی شده است، استفاده شده است. در این آزمون‌ها از جمع دو بردار با یکدیگر و یا جابه‌جایی بخشی از حافظه استفاده می‌شود. نتایج این آزمون‌ها در جدول ۲ نمایش داده شده است. سرعت حافظه اصلی سیستم شماره ۱ و ۲ تقریباً با یکدیگر مساوی و بسیار کمتر از سیستم شماره ۳ است، در حالی‌که حافظه‌های اصلی پردازنده‌های گرافیکی سیستم‌های شماره ۱ و ۳ از سرعت به مراتب بالاتری نسبت به پردازنده گرافیکی سیستم شماره ۲ برخوردارند. این تفاوت‌ها نشان

جدول ۲- سرعت حافظه سیستم‌های رایانه‌ای مورد استفاده برحسب GB/s

ردیف	پردازنده		پردازنده گرافیکی	
	Add	Copy	Add	Copy
۱	۵/۳	۴/۹	۳۶/۵	۱۲۱/۰
۲	۵/۲	۴/۶	۱۳/۱	۲۶/۰
۳	۱۴/۲	۱۳/۱	۶۵/۳	۱۳۴/۰

جدول ۳- مشخصات ماتریس‌ها

ماتریس	تعداد سطرها	تعداد ستون‌ها	تعداد عناصر غیر صفر	متوسط تعداد عناصر غیر صفر در هر سطر	نام ماتریس
Protein	۳۶۴۱۷	۳۶۴۱۷	۴۳۴۴۷۶۵	۱۱۹	pdb1HYS.mtx
FEM / Spheres	۸۳۳۳۴	۸۳۳۳۴	۶۰۱۰۴۸۰	۷۲	consph.mtx
FEM / Cantilever	۶۲۴۵۱	۶۲۴۵۱	۴۰۰۷۳۸۳	۶۴	cant.mtx
Wind Tunnel	۲۱۷۹۱۸	۲۱۷۹۱۸	۱۱۶۳۴۴۲۴	۵۳	pwtk.mtx
FEM / Ship	۱۴۰۸۷۴	۱۴۰۸۷۴	۷۸۱۳۴۰۴	۵۵	shipsec1.mtx

می‌دهد سیستم‌های مورد استفاده، دارای مشخصات متنوعی از نظر پردازنده و پردازنده گرافیکی هستند؛ که این امر باعث می‌شود نتایج حاصله از این تحقیق در شرایط مختلف قابل استناد باشد.

برای بررسی کارایی روش پیشنهادی، بخشی از یک سری ماتریس‌های استاندارد مورد استفاده در مرجع [۶] که متقارن و مثبت معین بودند، درنظر گرفته شدند. برخی از خصوصیات دیگر این ماتریس‌ها در جدول ۳ نشان داده شده است. هم‌چنین توضیحات بیشتر در مورد این ماتریس‌ها در مرجع [۱۹] آمده است. برای مقایسه کارایی روش‌های ارائه شده در این تحقیق، علاوه بر دو شکل پیشنهادی از روش گرادیان مزدوج، حل دستگاه‌های معادلات خطی توسط پردازنده و نیز توسط پردازنده گرافیکی هر دو توسط کتابخانه توابع وینا سی ال انجام شده است. از آنجا که تمام روش‌های مورد استفاده از نظر همگرایی با یکدیگر معادل بوده و تنها به‌علت خطاهای گرد کردن تعداد تکرارهای آنها با یکدیگر متفاوت است، برای مقایسه نسبی کارایی روش‌ها با یکدیگر زمان مورد نیاز برای ۱۰۰۰ تکرار هر روش درنظر گرفته می‌شود. شرط همگرایی به‌نحوی درنظر گرفته شده است که با انجام این تعداد تکرار حل خاتمه نیابد.

زمان اجرا در هر مورد و نسبت افزایش سرعت نسبت به حل انجام شده توسط کتابخانه توابع وینا سی ال روی پردازنده در جدول‌های ۴ تا ۶ آمده است. بررسی نتایج نشان می‌دهد روش پیشنهادی روی سیستم‌های مختلف بین ۵ تا ۱۵ برابر سریع‌تر از کتابخانه توابع وینا سی ال روی پردازنده حل دستگاه معادلات را به انجام رسانده است. در سیستم شماره ۲ که حافظه اصلی پردازنده گرافیکی به نسبت بسیار کندتر از سایر سیستم‌ها است، حل دستگاه معادلات توسط پردازنده گرافیکی با کمک کتابخانه توابع وینا سی ال کندتر از حل دستگاه توسط همین کتابخانه روی پردازنده اصلی است. این مورد به روشنی نشان می‌دهد که به‌علت استفاده این کتابخانه از شکل ساده ضرب ماتریس تنک در بردار توسط کتابخانه

جدول ۴- زمان اجرا برحسب ثانیه و نسبت افزایش سرعت روی سیستم شماره ۱

ماتریس	CPU (ViennaCL)	GPU (ViennaCL)		GPU Proposed (#1)		GPU Proposed (#2)	
	زمان اجرا	زمان اجرا	سرعت نسبی	زمان اجرا	سرعت نسبی	زمان اجرا	سرعت نسبی
Protein	۱۶/۳۰	۶/۲۵	۲/۶۱	۱/۶۷	۹/۷۶	۱/۶۷	۹/۷۴
FEM / Spheres	۲۶/۳۵	۷/۱۵	۳/۶۹	۲/۲۴	۱۱/۷۴	۲/۲۵	۱۱/۷۲
FEM / Cantilever	۱۸/۱۶	۴/۹۹	۳/۶۴	۱/۶۸	۱۰/۸۲	۱/۶۸	۱۰/۷۸
Wind Tunnel	۵۸/۶۵	۱۲/۰۹	۴/۸۵	۳/۷۸	۱۵/۵۰	۳/۷۹	۱۵/۴۶
FEM / Ship	۳۷/۵۷	۹/۲۱	۴/۰۸	۲/۵۷	۱۴/۶۴	۲/۵۷	۱۴/۶۱

جدول ۵- زمان اجرا برحسب ثانیه و نسبت افزایش سرعت روی سیستم شماره ۲

ماتریس	CPU (ViennaCL)	GPU (ViennaCL)		GPU Proposed (#1)		GPU Proposed (#2)	
	زمان اجرا	زمان اجرا	سرعت نسبی	زمان اجرا	سرعت نسبی	زمان اجرا	سرعت نسبی
Protein	۱۶/۰۵	۲۴/۰۵	۰/۶۷	۲/۶۱	۶/۱۶	۲/۶۰	۶/۱۶
FEM / Spheres	۲۳/۱۱	۳۴/۷۱	۰/۶۷	۳/۹۸	۵/۸۱	۴/۰۳	۵/۷۴
FEM / Cantilever	۱۶/۰۲	۲۲/۴۷	۰/۷۱	۲/۶۱	۶/۱۵	۲/۶۰	۶/۱۶
Wind Tunnel	۴۷/۴۹	۶۴/۹۰	۰/۷۳	۷/۵۶	۶/۲۸	۷/۵۷	۶/۲۷
FEM / Ship	۳۲/۰۲	۴۲/۴۹	۰/۷۵	۵/۱۹	۶/۱۷	۵/۱۹	۶/۱۷

جدول ۶- زمان اجرا برحسب ثانیه و نسبت افزایش سرعت روی سیستم شماره ۳

ماتریس	CPU (ViennaCL)	GPU (ViennaCL)		GPU Proposed (#1)		GPU Proposed (#2)	
	زمان اجرا	زمان اجرا	سرعت نسبی	زمان اجرا	سرعت نسبی	زمان اجرا	سرعت نسبی
Protein	۸/۸۴	۷/۶۰	۱/۱۶	۰/۹۹	۸/۸۹	۰/۹۰	۹/۸۰
FEM / Spheres	۱۱/۳۵	۱۰/۳۵	۱/۱۰	۱/۴۸	۷/۶۶	۱/۴۱	۸/۰۴
FEM / Cantilever	۷/۰۲	۷/۱۱	۰/۹۹	۱/۰۸	۶/۵۲	۱/۰۲	۶/۹۱
Wind Tunnel	۲۱/۱۲	۱۸/۶۰	۱/۱۴	۲/۶۴	۷/۹۸	۲/۶۱	۸/۱۰
FEM / Ship	۱۴/۰۸	۱۲/۸۳	۱/۱۰	۲/۰۱	۷/۰۲	۱/۹۱	۷/۳۷

توابع وینا سی ال [۶]، دسترسی به حافظه گرافیکی بسیار نامنظم‌تر است که این مورد افت بسیار شدید سرعت حل را در پی دارد. در همین سیستم روش پیشنهادی به‌علت استفاده از یک الگوریتم با کارایی بالا برای ضرب ماتریس تنک در بردار که توسط نگارنده پیشنهاد شده است، توانسته است حل دستگاه معادلات را ۵ تا ۶ برابر سریع‌تر از کتابخانه توابع وینا سی ال روی پردازنده اصلی و ۸ تا ۹ برابر سریع‌تر از همین کتابخانه روی پردازنده گرافیکی انجام دهد. نکته قابل ذکر دیگر در نتایج به‌دست آمده، کارایی بالای روش پیشنهادی در مقابل پردازنده قوی مورد استفاده در سیستم شماره ۳ است. از سوی

دیگر کارایی هر دو شکل پیاده‌سازی شده از روش گرادیان مزدوج تقریباً در یک سطح است که علت این موضوع را می‌توان در نقش بسیار عمده هسته محاسباتی ضرب ماتریس‌های تنک در بردار، پیشرفت بسترهای نرم‌افزاری زبان آزاد محاسباتی و در نتیجه کاهش زمان اضافی برای فراخوانی هسته‌های محاسباتی و نیز افزایش کارایی کامپایلرهای این زبان در بهینه‌سازی کد نهایی جستجو کرد.

۷- نتیجه‌گیری

در این مقاله با استفاده از یک الگوریتم با کارایی بالا برای ضرب ماتریس تنک در بردار که توسط نگارنده پیشنهاد شده است [۶]، روشی برای حل دستگاه‌های معادلات خطی بر مبنای روش گرادیان مزدوج ارائه گردید. برای دستیابی به کارایی بالاتر، دو شکل از این روش که به‌نظر می‌رسید بیشترین سازگاری را با مدل برنامه‌نویسی زبان آزاد دارند، درنظر گرفته شده و پیاده‌سازی شدند. نتایج به‌دست آمده روی سیستم‌های نسبتاً متنوع مورد بررسی، نشان می‌دهد روش پیشنهادی قادر است دستگاه‌های معادلات خطی را ۵ تا ۱۵ برابر سریع‌تر از کتابخانه توابع وینا سی ال روی پردازنده اصلی و ۳ تا ۹ برابر سریع‌تر از همین کتابخانه روی پردازنده گرافیکی حل نماید.

واژه نامه

1. direct methods
2. iterative methods
3. conjugate gradient
4. bi-conjugate gradient
5. Krylov sub-space methods
6. symmetric positive definite
7. bi-conjugate gradient stabilized
8. high performance computing (hpc)
9. local area network (lan)
10. distributed memory
11. multi-core
12. shared memory
13. computational accelerator
14. graphics processing unit (gpu)
15. general processing on graphics processing units (gpgpu)
16. cuda
17. NVIDIA
18. Apple
19. open computing language (opencl)
20. open standard
21. compute kernel
22. sparse
23. reduction
24. parallel reduction
25. workgroup
26. open multi-processing (openmp)
27. pre-processor macros
28. time marching
29. vector update
30. kernel fusing
31. STREAM

مراجع

1. Saad, Y., “Iterative Methods for Sparse Linear Systems”, 2nd Edition, SIAM, 2003.
2. Van Der Vorst, H. A., “Iterative Krylov Methods for Large Linear Systems”, Cambridge University Press, 2003.
3. Pasenau, M. and Jiménez, A. F., “Implementación de Algoritmos Numéricos en una Tarjeta Gráfica”, *Centro Internacional de Métodos Numéricos en Ingeniería (CIMNE)*, 2006.
4. NVIDIA CUDA, http://www.nvidia.com/object/cuda_home_new.html.
5. OpenCL, http://www.khronos.org/opencl.
۶. مسیبی، ف.، "پیاده‌سازی یک هسته قابل تنظیم ضرب ماتریس‌های تنک در بردار به کمک زبان آزاد محاسباتی روی پردازنده‌های گرافیکی"، *روش‌های عددی در مهندسی*، سال ۳۲، شماره ۱، تابستان ۱۳۹۲.
7. AMD Accelerated Parallel Processing OpenCL, AMD, 2011.
8. NVIDIA CUDA C Programming Guide, NVIDIA, 2012.
9. Hestenes, M. R., and Stiefel, E., “Methods of Conjugate Gradients for Solving Linear Systems”, *Journal of Research of the National Bureau of Standards*, Vol. 49, No. 6, 1952.
10. Shewchuk, J. R., “An Introduction to the Conjugate Gradient Method Without the Agonizing Pain”, Edition 1¼, Carnegie Mellon University, 1994.
11. Ament, M., Knittel, G., Weiskopf, D. and Strasser, W., “A Parallel Preconditioned Conjugate Gradient Solver for the Poisson Problem on a Multi-GPU Platform”, *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pp. 583-592, IEEE Computer Society Washington, USA, 2010.
12. Georgescu, S., and Okuda, H., “Conjugate Gradients

on multiple GPUs", *International Journal for Numerical Methods in Fluids*, Vol. 64, pp. 1254-1273, 2010.
13. Mehri Dehnavi, M., Fernández, D. M., and Giannacopoulos, D., "Enhancing the Performance of Conjugate Gradient", *IEEE Transaction on Magnetics*, Vol. 47, No. 5, pp. 1162-1165, 2011.
14. Cevahir, A., Nukada, A., and Matsuoka, S., "Fast Conjugate Gradients with Multiple GPUs", *Proceedings of the 9th International Conference on Computational Science*, Part I, pp. 893-903, Springer-Verlag, Berlin, Heidelberg, 2009.
15. ViennaCL project homepage, http://viennacl.sourceforge.net.
16. Fernández, D. M., Giannacopoulos, D., and Gross, W., "Multicore Acceleration of CG Algorithms Using Blocked-Pipeline-Matching Techniques", *IEEE Transactions on Magnetics*, Vol. 46, No. 8, pp. 3057-3060, 2010.
17. Chronopoulos, A. T., and Gear, C. W., "S-Step Iterative Methods for Symmetric Linear Systems", *Journal of Computational and Applied Mathematics*, Vol. 25, pp. 153-168, 1989.
18. McCalpin, J. D., "Memory Bandwidth and Machine Balance in Current High Performance Computers", *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19-25, 1995.
19. Williams, S., Oliker, L., Vuduc, R., Shalf, J., Yelick, K., and Demmel, J., "Optimization of Sparse Matrix-vector Multiplication on Emerging Multicore Platforms", *Proceedings the 2007 ACM/IEEE Conference on Supercomputing*, ACM New York, NY, USA, 2007.