دانشگاه فردوسی مشهد
دانشکده مهندسی - گروه کامپیوتر
آزمایشگاه بینایی ماشین

برنامه‌نویسی موازی در مطلب

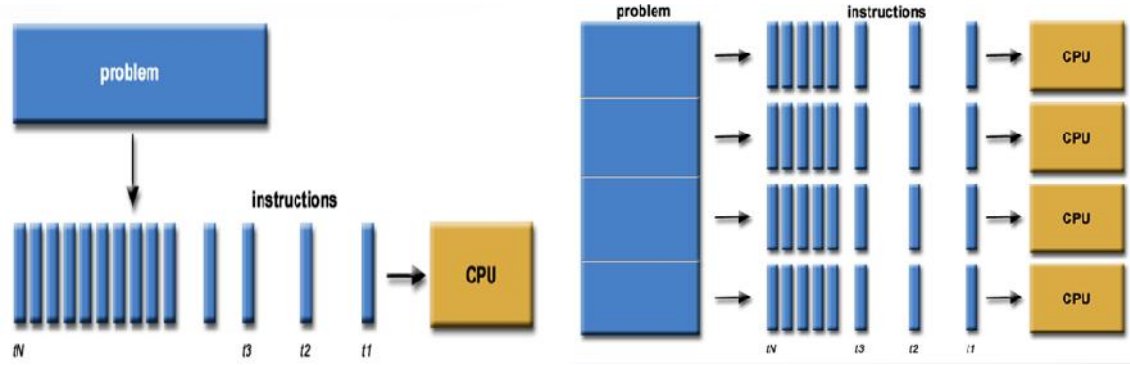
(سیستم‌های Desktop چند هسته‌ای)

MV Lab
آزمایشگاه بینایی ماشین
گروه مهندسی کامپیوتر
دانشکده مهندسی
دانشگاه فردوسی مشهد
تلفن ۰۶ ۳۳ ۸۷۶ ۰۵۱۱

نسخه	تهیه کننده	تاریخ	اصلاحات
1.0	حجت باقرزاده حمیدرضا پوررضا	پاییز ۹۲	نسخه‌ی اولیه
1.1	حجت باقرزاده حمیدرضا پوررضا	زمستان ۹۲	تکمیل

۱- برنامه نویسی موازی چیست؟

بطور معمول برنامه ها بصورت سریال نوشته می‌شوند. بگونه ای که یک وظیفه بعد از اتمام وظیفه دیگر اجرا شده و اینکار تا زمانیکه کل کد خاتمه می‌یابد، ادامه دارد. نکته اینجاست که در حالت در هر بار فقط یک وظیفه اجرا می‌شود. با توجه به این تعریف، استفاده از واحدهای پردازش چندتایی برای حل یک مسئله در یک زمان بمعنی محاسبات موازی می‌باشد. بطورساده محاسبات موازی بمعنی استفاده همزمان از منابع محاسباتی متعدد برای حل مسئله می‌باشد. برای اینکار مسئله به قسمت های جداگانه ای شکسته شده که می‌توانند بصورت همزمان اجرا شوند. در ادامه هر قسمت به یک سری از دستورات شکسته شده و این دستورات روی CPU های متفاوت اجرا می‌گردند.



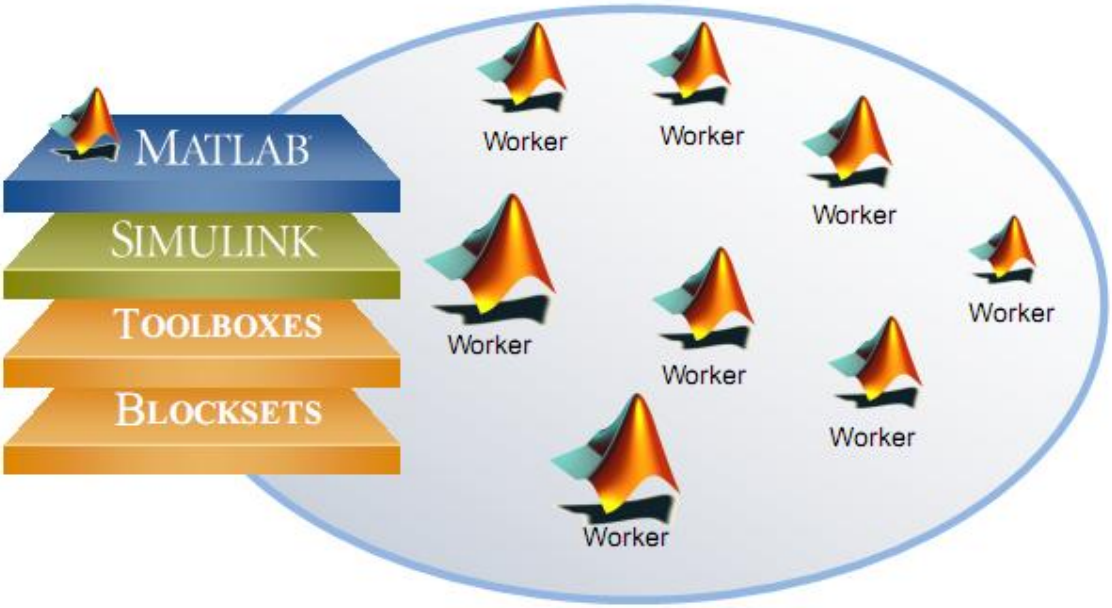
محاسبات سری　　محاسیات موازی

۲- برنامه نویسی موازی در مطلب:

برنامه موازی در مطلب (Parallel Matlab) یک توسعه در نرم افزار مطلب می‌باشد که از فواید کامپیوترهای چندهسته ای و سیستم های توزیع شده بدرستی بهره برده است. ابزار برنامه محاسبات موازی (Parallel Computing Toolbox یا PCT) روی یک سیستم desktop اجرا شده و می‌تواند باتوجه به مشخصات سخت افزاری سیستم تا ۸ یا ۱۲ هسته را زیربار ببرد.

کاربر می‌تواند از سه طریق زیر برنامه را بصورت موازی اجرا نماید:

- دستوراتی را تایپ نماید که به صورت موازی اجرا شوند.
- یک M-file فایلی را فراخوانی کند که بصورت موازی اجرا شود.
- یک M-file فایل را برای اجرا به batch ارسال نماید. (نکته اینکه در این حالت اجرا بصورت برهم کنش interactive نخواهد بود و در پس زمینه background اجرا می‌شود).



سرور محاسبات توزیع شده (Distributed Computing Server) اجراهای موازی را در مطلب روی یک Cluster با ده ها یا صدها هسته کنترل می‌کند. با استفاده از یک cluster که مطلب را بصورت موازی اجرا می‌کند. کاربر قابلیت های زیر را دارد.

۱. یک M-file فایل را از یک کامپیوتر Desktop برای اجرا روی Cluster ارسال نماید.
۲. به سیستم cluster وارد شده و بصورت برهم کنشی interactive برنامه خود را اجرا نماید.
۳. به سیستم cluster وارد شده و M-file فایل را برای اجرا با batch ارسال نماید.

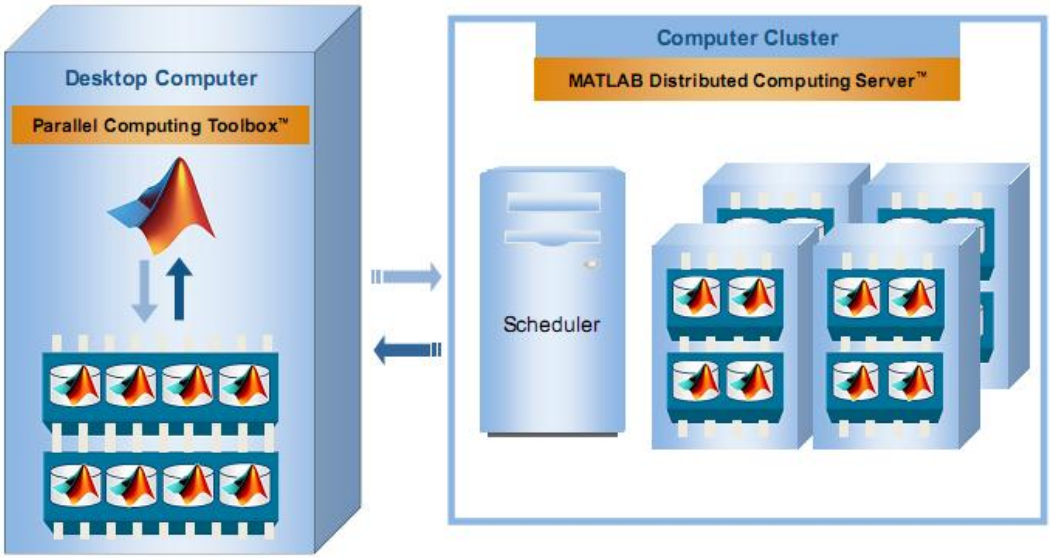
انتخاب های ۱ و ۳ به کاربر این اجازه را می‌دهد که از نرم افزار در سیستم desktop خود و یا cluster خارج شده و در زمان دیگری برای بررسی پایان یافتن محاسبات بازگردد. برای مثال cluster آزمایشگاه تخصصی اجازه اجرای همزمان برنامه موازی مطلب را روی ۹۶ هسته پردازنده می‌دهد.

روش های اجرای یک برنامه موازی مطلب به شرح زیر می‌باشد:

- استفاده از matlabpool که بصورت برهم کنش محلی interactive localمی‌باشد و برای سیستم های desktop مناسب است.
- استفاده از batch یا createTask بصورت محلی غیرمستقیم indirect localمی‌باشد.

- استفاده از batch یا createTask بصورت غیرمحلی غیرمستقیم indirect remoteمی‌باشد. و نیازمند راه اندازی می‌باشد.

یک clusterمی‌تواند وظیفه های موازی مطلب را که از desktop های کاربران ارسال می‌شود، می‌پذیرد. و نتایج را زمانی که اجرای وظیفه تمام شود، بازمی‌گرداند. برای اجرای این حالت بایستی یک بار سیستم کاربر تنظیم شده که بداند چگونه با یک cluster ارتباط برقرار کند و اینکه بداند چگونه با آن نسخه کپی از نرم افزار مطلب روی cluster متصل شود.



از جمله راه های نوشتن یک برنامه موازی در مطلب می‌توان به موارد زیر اشاره کرد:

- **حلقه Parfor** :حلقه های for که بصورت مناسب طراحی شده باشند می‌توانند به حلقه های parfor تبدیل شوند.دستور parfor یک روش ساده از ساختن حلقه های For برای اجرای موازی می‌باشد و همانند OpenMP عمل می‌نماید.
- **دستور spmd** : این دستور همکاری پروسس ها را همگام سازی (synchronize) می‌نماید. (این دستور در مطلب ورژن ۲۰۱۱ و بعد از در نرم افزار مطلب قرار داده شده است.)دستور spmd به کاربر اجازه می‌دهد تقریبا هر مدل محاسباتی موازی را طراحی کند. این ابزار برای اجرای موازی بسیار قوی می‌باشد، اما نیازمند این است که روال اجرای برنامه و داده ها بطور مناسب طراحی گردد.
- **استفاده از Graphic Process Unit (GPU)**: ابزار Parallel Computing Toolbox در مطلب GPUArray را ارائه می‌نماید. GPUArray یک آرایه خاص با توابع زیادی می‌باشد که اجازه می‌دهد اجرای محاسبات بر روی CUDA-enabled NVIDIA GPU مستقیما از طریق مطلب اجرا شود.

۲-۱- حلقه parfor:

۲-۱-۱- مثالی از نحوه استفاده از حلقه parfor بجای حلقه for:

```
function q=quad_fun(n,a,b)
    q=0.0;
    w=(b-a)/n;
for i=1:n
        x=((n-i)*a+(i-1)*b)/(n-1);
fx=bessely(4.5,x);
        q=q+w*fx;
end
return
end
```

تابع quad_fun انتگرال یک تابع خاص را در بازه [a,b] تخمین می‌زند. این تابع تخمین را با ارزیابی تابع در n فاصله برابر، با استفاده از ضرب هر مقدار در وزن (b-a)/n انجام می‌دهد. این مقادیر کمی‌می‌تواند بعنوان مساحت مستطیل های کوچک که در زیر منحنی ها قرار دارند، در نظر گرفته شوند. و مجموع آنها یک تخمین برای کل مساحت زیر منحنی از a به b می‌باشد. ما می‌توانیم این ریزمساحت ها رابه هر ترتیبی حساب کنیم. ما حتی می‌توانیم این ریزمساحت ها را بصورت همزمان محاسبه کنیم. فرض کنید روش هایی برای ذخیره نتایج جزئی و جمع آنها با یکدیگر به روش سازمان دهی شده، وجود دارد.

```
function q=quad_fun(n,a,b)
    q=0.0;
    w=(b-a)/n;
parfor i=1:n
        x=((n-i)*a+(i-1)*b)/(n-1);
fx=bessely(4.5,x);
        q=q+w*fx;
end
return
end
```

نسخه موازی از تابع quad_fun هم محاسبات مشابه ای را انجام می‌دهد. دستور parfor چگونگی محاسبه را در برنامه تغییر می‌دهد. ادعا می‌شود که تمامی‌تکرارهای حلقه بصورت مستقل هستند. و می‌توانند به هر ترتیبی اجرا شوند و یا بصورت موازی اجرا شوند. اجرا با یک تک پروسس client شروع می‌گردد. و زمانیکه به اجرای حلقه parforمی‌رسیم، یک گروه pool از انجام دهنده ها Workers بکار گرفته می‌شوند. به هر worker تعدادی از تکرارهای حلقه انتساب داده می‌شود. زمانیکه کل حلقه پایان می‌یابد، پروسس client کنترل اجرا را در دست می‌گیرد. نرم افزار مطلب اطمینان می‌دهد که نتایج همیشه یکسان است، چه برنامه بصورت ترتیبی اجرا گردد و با استفاده از گروهی از انجام دهنده worker ها. کاربر می‌تواند برای تعیین تعداد انجام دهنده ها workers تا زمان اجرا صبر نماید. (یعنی تعداد آنها را در زمان اجرا مشخص نماید.)

شرایط زیر برای اجرای دستور parfor الزامی‌است:

- سیستم شما بایستی پروسس چند هسته ای داشته باشد و یا چند پروسس داشته باشد.
- نسخه نرم افزار مطلب شما بایستی حداقل 2008a باشد.
- بایستی ابزار محاسبات موازی Parallel Computing Toolbox نصب شده باشد.

انجام دهنده ها workers با استفاده از دستور matlabpool فراخوانی می‌شوند. برای اجرای تابع quad_fun به روش زیر عمل می‌کنیم :

```
n = 10000; a = 0; b = 1;
matlabpool open local 4
```

```
q = quad_fun(n, a, b);
matlabpool close
```

کلمه local برای انتخاب تنظیمات محلی می‌باشد. بدین معنی که هسته های انتساب داده شده به انجام دهنده ها workers بایستی روی خود همان کامپیوتر باشند. مقدار ۴ تعداد انجام دهنده هایی workers است که شما درخواست می‌کنید. این تعداد در سیستم های محلی می‌تواند تا ۱۲ (در نسخه 2013a) افزایش یابد. نیازی نیست این تعداد با تعداد هسته ها برابر باشد.

نکته : اگر با خطای زیر در اجرای دستور matlabpool opne local 4 مواجه شدید:

You requested a minimum of 4 workers, but the cluster "local" has the NumWorkers property set to allow a maximum of 2 workers. To run a communicating job on more workers than this (up to a maximum of 12 for the Local cluster), increase the value of the NumWorkers property for the cluster. The default value of NumWorkers for a Local cluster is the number of cores on the local machine.

برای افزایش تعداد انجام دهنده ها NumWorkersمی‌توان به یکی از روش های عمل کرد. در نظر داشته باشید که استفاده از تعداد انجام دهنده های worker بیشتر، ممکن است کارایی را نسبت به زمانیکه تعداد انجام دهنده worker ها با تعداد هسته ها برابر است، کاهش دهد

- پنجرهCluster Profile Manager(Parallel->Manage Cluster Profiles) را باز کنید. پروفایل local را انتخاب کرده و کلید ویرایش را فشاردهید و مقدار NumWorker را افزایش دهید. این مقدار می‌تواند تا ۱۲ افزایش یابد و می‌تواند از تعداد هسته های پردازنده بیشتر باشد.
- می‌توان این تغییرات بصورت کد نیز انجام داد.

```
myCluster = parcluster('local');
myCluster.NumWorkers = 4; % 'Modified' property now TRUE
saveProfile(myCluster);   % 'local' profile now updated,
                          % 'Modified' property now FALSE
```

اجرای بصورت غیر مستقیم نیاز به یک فایل M-file به شرح زیر می‌باشد.

```
n = 10000; a = 0; b = 1;
q = quad_fun(n, a, b);
```

حال اطلاعات مورد نیاز برای اجرای script را تعریف می‌کنیم.

```
job = batch( 'quad_script','matlabpool', 4,'Configuration', 'local','FileDependencies',{ 'quad_fun' })
```

دستور batchمی‌تواند این کار job درخواستی را به هرجایی ارسال کند و نتایج را دریافت نماید. درصورتیکه یک حساب روی سیستم مورد نظر ساخته شده با شد و یک تنظیمات configuration نیز روی سیستم Desktop برای توصیف نحوه دسترسی به سیستم دیگر ، تعریف شده باشد. برای مثال در آزمایشگاه تخصصی Virginia، یک سیستم Desktopمی‌تواند یک batch job را به cluster ارسال کند و 32 هسته را درخواست نماید.

```
job=batch('quad_script','matlabpool',32,'Configuration','ithaca_2011b','FileDependencies'
,{'quad_fun'})
```

به هر دو صورت محلی و از راه دور، دستورات زیر کار job را برای اجرا ارسال می‌کند، و تا پایان آن منتظر می‌ماند و سپس نتایج را در فضای کاری مطلب بارگذاری می‌کند.

```
job=batch( ...informationaboutjob... )
```

```
submit(job);
wait(job);
load(job);
```

برای انجام این حالت نیاز است که شما در سیستم logged in باشید تا مقادیر job بتواند برای تعیین هویت کردن خروجی ها در دستور load() استفاده شود.

بهتر است بدانید که می‌تواند بعد از دستور submit() از سیستم خارج شوید. و در زمان ورود مجدد بایستی مولفه job را بازیابی کنید و سپس نتایج را بارگذاری نمایید.

```
job = batch( ...informationaboutjob... )
submit(job);
Exit MATLAB, turn off machine, go home.
Comeback, restart machine, start MATLAB:
sched = findResource();
job = findJob(sched);
load(job);
```

۲-۱-۲- مثال اعداد اول برای حلقه parfor:

در این مثال ما یک محاسبه ساده که شامل حلقه می‌باشد را بررسی می‌کنیم. این برنامه تعیین می‌کند چه تعداد اعداد اول بین 1 و N وجود دارد. برای اجرای بزرگتر برنامه متغیر N را افزایش می‌دهیم. برای مثال اگر مقدار N را دوبار کنیم، زمان اجرای برنامه ۴ برابر می‌شود. تابع prime(n) در حالت ترتیبی به فرم زیر می‌باشد.

```
function total = prime(n)
%%PRIMEreturnsthenumberofprimesbetween1andN.
total = 0;
for i = 2:n
prime = 1;
for j = 2:i-1
if (mod(i, j) == 0)
prime = 0;
end
end
total = total + prime;
end
return
end
```

حلقه ای را که ایندکس شمارنده آن iمی‌باشد را با تبدیل for به parfor موازی می‌کنیم. محاسبات مقادیر محتلف i مستقل از یکدیگر می‌باشند. فقط یک متغیر total وجود دارد که مستقل نمی‌باشد. این متغیر محاسبه جمع ساده می‌باشد و ما فقط به نتیجه نهایی آن نیاز داریم. نرم افزار مطلب هوشمندی کنترل جمع در حالت موازی را دارد. در نتیجه برای تبدیل این برنامه به حالت موازی این حلقه for را با parfor تعویض می‌کنیم.

```
function total = prime(n)
%%PRIMEreturnsthenumberofprimesbetween1andN.
total = 0;
parfor i = 2:n
prime = 1;
for j = 2:i-1
if (mod(i, j) == 0)
prime = 0;
end
end
total = total + prime;
end
return
end
```

برنامه را با تعداد انجام دهنده worker های متقاوت و بزرگی متفاوت اجرا کرده و زمان های زیر را بدست آمده است.

Run PRIME_PARFOR with 0, 1, 2, and 4 workers. Time is measured in seconds.				
N	1+0	1+1	1+2	1+4
50	0.067	0.179	0.176	0.278
500	0.008	0.023	0.027	0.032
5000	0.100	0.142	0.097	0.061
50000	7.694	9.811	5.351	2.719
500000	609.764	826.534	432.233	22.284

تناقض هایی که در این نتایج بدست آمده نشان دهنده نحوه استفاده درست از حالت برنامه نویسی موازی می‌باشد.
نکته های مهمی‌که در برنامه نویسی موازی بایستی درنظر گرفت به قرار زیر می‌باشد :

- موازی سازی زمانی ارزشمند می‌باشد که برنامه به حد کافی بزرگ باشد.
- موازی سازی زمانی ارزشمند است که تعداد انجام دهنده ها worker زیاد باشد.

۲-۱-۳- نکات مهم در رابطه با نجوه استفاده از دستور parfor:

- ساده ترین راه موازی سازی یک برنامه روی حلقه های for برنامه تمرکز دارد. سوال اینست که آیا تکرارهای این حلقه می‌تواند به هر ترتیبی اجرا شود، بدون اینکه روی نتیجه نهایی تاثیر گذارد؟ اگر جواب به این سوال 'بله' باشد، این حلقه می‌تواند بصورت موازی طراحی گردد.
- اگر حلقه های تو در تو در برنامه وجود دارد، بطور کلی نمی‌توان کلیه حلقه ها را با حلقه های parfor تو در تو جایگذاری نمود. اگر حلقه خارجی را بتوان بصورت موازی طراحی کرد، پس این حلقه همانی است که بایستی با parfor طراحی شود. اگر حلقه خارجی را نتوان بصورت موازی طراحی نمود، می‌توان حلقه های داخلی تر را بصورت موازی طراحی نمود.
- امن ترین فرض برای یک حلقه parfor این است که هر تکرار از حلقه توسط یک انجام دهنده worker مجزا اجرا و بررسی می‌گردد. اگر یک حلقه for وجود دارد که تمامی‌تکرارها بصورت کاملا مستقل از یکدیگر هستند، در نتیجه این حلقه یک کاندید خوب برای تبدیل به حلقه parforمی‌باشد.
- بدنه حلقه parfor بایستی کاملاً شفاف و واضح باشد. بدین معنی که تمامی‌ارجاع ها به متغیرها بایستی قابل دیدن visible باشد.
- بدنه حلقه parfor نمی‌تواند شامل دستور spmd باشد و دستور spmd نیز نمی‌تواند شامل حلقه parfor باشد.
- بدنه حلقه parfor نمی‌تواند شامل دستور break یا return باشد.
- بدنه حلقه parfor نمی‌تواند شامل تعریف متغیر جامع global و یا persistent باشد.
- تغییرات انجام شده روی handle کلاس ها که روی انجام دهنده ها workers در طول اجرای حلقه انجام می‌گردد، بصورت اتومات به پروسس client انتشار نمی‌یابد.

- شما می‌توانید یک فایل M-file دیگر را در حلقه parfor فراخوانی کنید، ولی آن فایل نمی‌تواند شامل حلقه parfor باشد.

۳- نحوه استفاده از SPMD:

دستور spmd یک نسخه ساده شده از MPIمی‌باشد. در این مدل، یک پروسس client وجود دارد که تمامی‌انجام دهنده workers ها را که روی یک تک برنامه همکاری می‌کنند، کنترل می‌کند. هر انجام دهنده (که گاهی اوقات یک lab نامیده می‌شود.) یک شناسه مجزا دارد، و می‌داند که چه تعداد انجام دهنده worker وجود دارد و می‌تواند رفتارش را براساس همان شناسه تعیین کند.

- هر انجام دهنده worker روی یک هسته مجزا اجرا می‌شود. (بصورت ایده آل)
- هر انجام دهنده worker از یک فضای کاری workspace مجزا استفاده می‌کند.
- یک برنامه مشترک در بین آنها استفاده می‌شود.
- انجام دهنده ها در نقاط همگام سازی synchronization از لحاظ اجرایی بایکدیگر ملاقات می‌کنند.
- برنامه clientمی‌تواند داده های هر انجام دهنده worker را تغییر دهد و یا آزمایش کند.
- هر دو انجام دهنده workerمی‌توانند از طریق سیستم پیغام با یکدیگر ارتباط برقرار کنند.

بصورت برهم کنشی interactive ما انجام دهنده worker را با استفاده از matlabpool درخواست می‌کنیم.

```
matlabpool open local 4
results = myfunc (args);
```

و یا با استفاده از batch برنامه را در پس زمینه background اجزا می‌کنیم.

```
job = batch ('myscript', 'matlabpool', 4, 'Configuration',  'local' )
```

نرم افزار مطلب در ابتدا یک انجام دهنده worker خاص را بنام client را ایجاد می‌کند. سپس تعداد انجام دهنده worker های خواسته شده را همراه با یک کپی از برنامه ایجاد می‌کند. هر انجام دهنده می‌داند که یک انجام دهنده workerمی‌باشد. و بایستی به دو تابع خاص دسترسی داشته باشد.

- تابع numlabs() : این تابع تعداد انجام دهنده ها worker ها را برمی‌گرداند.
- تابع labindex() : این تابع یک شناسه یکتا بین ۱ و numlabs() را به انجام دهنده ای که آنرا فراخوانی می‌کند بازمی‌گرداند.

معمولا پرانتز ها در برنامه ها نوشته نمی‌شود. ولی بیاد داشته باشید که اینها تابع هستند (هرچند بدون پرانتز) و متغیر نیستند. و نکته اینکه اگر پروسس client این توابع را فراخوانی کند، هر دوی آنها مقدار ۱ را باز می‌گردانند. این مقدار ۱ به این دلیل است که زمانیکه پروسس client درحال اجرا می‌باشد، انجام دهنده ها workers درحال اجرا نیستند. پروسس client تعداد انجام دهنده ها worker ها را تعیین می‌کند. تعداد انجام دهنده های workers قابل دسترس با دستور مشخص می‌گردد.

```
n = matlabpool ('size')
```

پروسس client و انجام دهنده ها workers یک تک برنامه مشترک دارند که تعدادی از دستورات در بلاک spmd محدود شده است. این بلاک با کلمه spmd شروع و با end خاتمه می‌یابد. پروسس client دستورات تا اولین بلاک spmd اجرا می‌کند. سپس این پروسس متوقف شده و انجام دهنده ها workers کد داخل بلاک را اجرا می‌کنند. هنگامیکه همه انجام

دهنده ها کارشان خاتمه یافت، پروسس client اجرای خود را ادامه می‌دهد. پروسس client و انجام دهنده ها فضاهای کاری workspace های مجزا در اختیار دارند، اما می‌توانند با یکدیگر ارتباط داشته و اطلاعات را انتقال دهند. متغیر های تعریف شده در قسمت client در انجام دهنده ها قابل دسترس می‌باشد، ولی قابل تغییر نیست. متغیرهای تعریف شده توسط انجام دهنده ها workers توسط پروسس client توسط یک قائده خاص هم قابل دسترس و هم قابل تغییر می‌باشد.

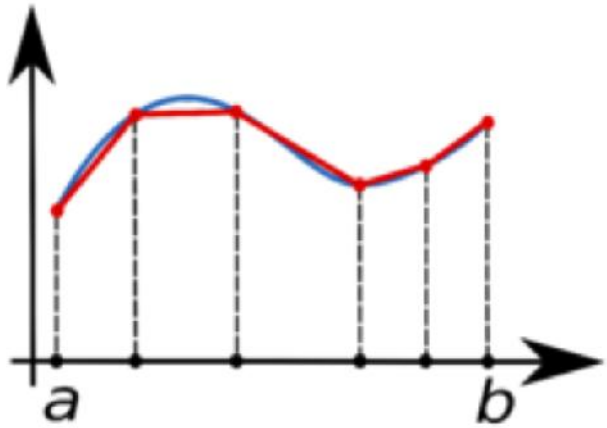
۳-۱- مثالی از نحوه کنترل فضاهای کاری در spmd به شرح زیر می‌باشد:

```
                      Client        Worker 1     Worker 2
                      a  b  e  |  c  d  f  |  c  d  f
                      ------------------------------
a = 3;                3  -  -  |  -  -  -  |  -  -  -
b = 4;                3  4  -  |  -  -  -  |  -  -  -
spmd                            |           |
  c = labindex();     3  4  -  |  1  -  -  |  2  -  -
  d = c + a;          3  4  -  |  1  4  -  |  2  5  -
end                             |           |
e = a + d{1};         3  4  7  |  1  4  -  |  2  5  -
c{2} = 5;             3  4  7  |  1  4  -  |  5  6  -
spmd                            |           |
  f = c * b;          3  4  7  |  1  4  4  |  5  6 20
end
```

یک برنامه می‌تواند شامل تعداد بیشماری بلاک spmd باشد. هنگامیکه اجرای یک بلاک spmd خاتمه می‌یابد، انجام دهنده ها متوقف می‌شوند، اما از بین نمی‌روند و فضاهای کاری آنها دست نخورده باقی می‌ماند. یک مجموعه متغیر که در یک بلاک spmd همان مقدار قبلی را در بلاک spmd بعدی نیز خواهد داشت.

در نرم افزار مطلب، متغیرهای تعریف شده در یک تابع زمانی از بین می‌روند که تابع خاتمه یابد. این قانون برای یک برنامه که یک تابع شامل بلاک spmd را فراخوانی می‌کند نیز درست است. تا زمانیکه اجرا در داخل تابع است، داده های انجام دهنده worker از یک بلاک spmd به بلاک دیگر spmd محافظت می‌شود، اما زمانیکه تابع خاتمه می‌یابد، داده های انجام دهنده worker همانند داده معمولی در مطلب، نیز از بین می‌روند.

۳-۲- قانون ذوزنقه – مثالی از نحوه استفاده spmd:



Area of one trapezoid = average height * base.

برای تخمین مساحت زیر منحنی با استفاده از یک ذوزنقه از فرمول زیر استفاده می‌کنیم.

$$\int_a^b f(x)\,dx \approx \left(\frac{1}{2}f(a) + \frac{1}{2}f(b)\right) * (b-a)$$

این تخمین را می‌توان با استفاده از n-1 ذوزنقه که توسط نقاط با فواصل مساوی تعریف می‌شود، بهبود بخشید.

$$\int_a^b f(x)\,dx \approx \left(\frac{1}{2}f(x_1) + f(x_2) + \ldots + f(x_{n-1}) + \frac{1}{2}f(x_n)\right) * \frac{b-a}{n-1}$$

اگر تعداد زیادی انجام دهنده worker در دسترس باشند، به هر کدام می‌توان یک قسمت از بازه را برای کار کردن را اختصاص داد و یم تخمین ذوزنقه را محاسبه کرد. با جمع کردن این تخمین ها می‌توان یک تقریب از انتگرال تابع روی کل بازه بدست آورد.

برای ساده سازی فرض می‌کنیم که بازه اولیه بین [0,1]می‌باشد و هر انجام دهنده یک a و b را برای نگهداری مقادیر انتهایی بازه اش تعریف می‌کند. اگر ۴ انجام دهنده worker داشته باشیم، بنابراین بازه [1/2,3/4] به انجام دهنده worker شماره ۳ اختصاص داده می‌شود. برای شروع برنامه هر انجام دهنده worker بازه خود را مشخص می‌کند.

```
spmd
        a = ( labindex - 1 ) / numlabs;
        b = labindex / numlabs;
end
```

هر انجام دهنده worker یک برنامه با فضای کاری مختص خودش می‌باشد. این انجام دهنده می‌تواند متغیرهای client را ببیند، اما معمولا نمی‌داند چه عملیاتی در سایر انجام دهنده ها اتفاق می‌افتد. هر انجام دهنده متغیرهای a و b را تعریف می‌کند، اما مقادیر مختلفی را در آن ذخیره می‌کند.

پروسس clientمی‌تواند فضاهای کاری همه انجام دهنده worker ها را مشاهده نماید. از آنجایی که مقادیر متعددی با استفاده از یک نام مشخص می‌شود، پروسس client بایستی ایندکس انجام دهنده worker را مقدار آنرا نیاز دارد، مشخص کند. بنابراین a{1} نحوه ارجاع به متغیر a را در انجام دهنده worker شماره ۱ مشخص می‌کند. پروسسclientمی‌تواند

این متغیر را بخواند و یا آنرا تغییر دهد. این متغیرها همانند سلول های آرایه می‌باشند. انجام دهنده ها workerمی‌توانند متغیرهای client را ببینند، از آنها کپی تهیه نمایند ولی نمی‌توانند داده client تغییر دهند.

بنابراین در QUAD هر انجام دهنده workerمی‌تواند a و b را چاپ نماید.

```
spmd
  a = ( labindex - 1 ) / numlabs;
  b =   labindex       / numlabs;
  fprintf ( 1, '  A = %f, B = %f\n', a, b );
end
```

و یا پروسس clientمی‌تواند همه آنها را چاپ کند.

```
spmd
  a = ( labindex - 1 ) / numlabs;
  b =   labindex       / numlabs;
end
for i = 1 : 4   <-- "numlabs" wouldn't work here!
  fprintf ( 1, '  A = %f, B = %f\n', a{i}, b{i} )
end
```

هر انجام دهنده workerمی‌تواند ذوزنقه خود را محاسبه نماید.

```
spmd
  x = linspace ( a, b, n );
  fx = f ( x );   <-- Assume f handles vector input.
  quad_part = ( b - a ) / ( n - 1 ) *
    * ( 0.5 * fx(1) + sum(fx(2:n-1)) + 0.5 * fx(n) );
  fprintf ( 1, '  Partial approx %f\n', quad_part );
end
```

نتایج زیر از اجرای برنامه بدست می‌آید :

```
2 Partial approx 0.874676
4 Partial approx 0.567588
1 Partial approx 0.979915
3 Partial approx 0.719414
```

ما در نهایت نیاز به یک پاسخ داریم، جمع این چهار تقریب. یک راه این است که پروسس client جمع همه جواب ها را بدست آورد.

```
quad = sum ( quad_part{1:4} );
fprintf (1, ' Approximation %f\n', quad);
```

و نتیجه نهایی به شرح زیر است:

```
Approximation 3.14159265
```

۳-۳- نکات مهم در رابطه با نحوه استفاده از بلاک دستوری spmd:

- نرم افزار مطلب بدنه دستورات spmd را روی انجام دهنده ها worker بصورت همزمان اجرا می‌کند.

- داخل بدنه دستورات spmd، هر انجام دهنده worker نرم افزار مطلب یک مقدار یکتا از labindex دارد، و numlabs تعداد کل انجام دهنده هایی که بلاک spmd را اجرا می‌کنند، دارد.
- در داخل بدنه دستورات spmd، تابع های ارتباطی برای کارهای موازی (مانند labSend و labReceive) می‌توانند داده ها بین انجام دهنده ها workers انتقال دهند.
- متغیرهای که مقادیر را از بدنه دستورات spmd بازمی‌گردانند، به شی های آرایه ای Composite Objects تبدیل می‌شوند.
- یک شی آرایه ای Compisite Object شامل ارجاعات به مقادیر ذخیره شده روی انجام دهنده های مطلب می‌باشد. و این مقادیر می‌توانند توسط یک ایندکس گذاری آرایه ای بازیابی شوند. داده واقعی انجام دهنده ها worker بصورت قابل دسترس برای اجراهای آینده spmdمی‌باشد، تازمانیکه شی های آرایه ای Composite ها روی پروسس client وجود داشته باشند و matlabpool هنوز بسته نشده باشد.
- قمست موازی کد با دستور spmd شروع می‌شود و با دستور end خاتمه می‌یابد. محاسبات در این بلاک ها در انجام دهنده های مطلب اتفاق می‌افتد. و زمانیکه این بلاک ها در حال اجرا هستند، پروسس client فعالیتی انجام نمی‌دهد.
- هر انجام دهنده worker به متغیر numlabs که شامل تعداد انجام دهنده ها workers است، دسترسی دارد، و هر انجام دهنده worker یک مقدار یکتا در متغیر labindex خودش دارد که بین ۱ و numlabsمی‌باشد.
- هر متغیر که توسط client تعریف شده باشد، توسط انجام دهنده ها workers قابل دیدن است. و می‌تواند در محاسبات داخل بلاک spmd استفاده گردد.
- هر متغیری که توسط انجام دهنده ها workers ها تعریف شده باشد، یک متغیر آرایه ای Compositeمی‌باشد. اگر متغیر X توسط انجام دهنده تعریف شده باشد، در نتیجه هر انجام دهنده worker مقدار خودش را دارد و یک مجموعه از مقادیر توسط client، با استفاده از ایندکس انجام دهنده worker قابل دسترس می‌باشد. بنابراین X{1} مقدار متغیر آرایه ای X در انجام دهنده ۱ می‌باشد.
- هر برنامه می‌تواند تعداد بیشماری بلاک spmd داشته باشد. زمانیکه برنامه اجرا یک بلاک spmd را به پایان رسانید، دستورات برنامه client اجرا می‌گردد. و در زمان ورود به بلاک spmd بعدی، تمام متغیرهایی که در بلاک spmd قبلی تعریف شده اند، در این بلاک نیز وجود خواهند داشت.
- انجام دهنده ها workers نمی‌توانند بطور مستقیم متغیرهای انجام دهنده های دیگر را ببینند. این ارتباطات از یک انجام دهنده به دیگری بایستی از طریق پروسس client صورت پذیرد.
- نکته اینکه عملیات های خاصی برای ترکیب متغیرها در بلاک های spmd وجود دارد. برای مثال دستور gplus مقادیر یک متغیر را که در تمام انجام دهنده ها workers وجود دارد باهم جمع می‌کند و به هر انجام دهنده نتیجه جمع را باز می‌گرداند.
- جنبه تک برنامگی single program از spmd یعنی اینکه یک کد یکسان در تمامیlab ها اجرا می‌شود. و زمانیکه اجرای بلاک spmd خاتمه می‌یابد، اجرای برنامه در پروسس client ادامه می‌یابد.

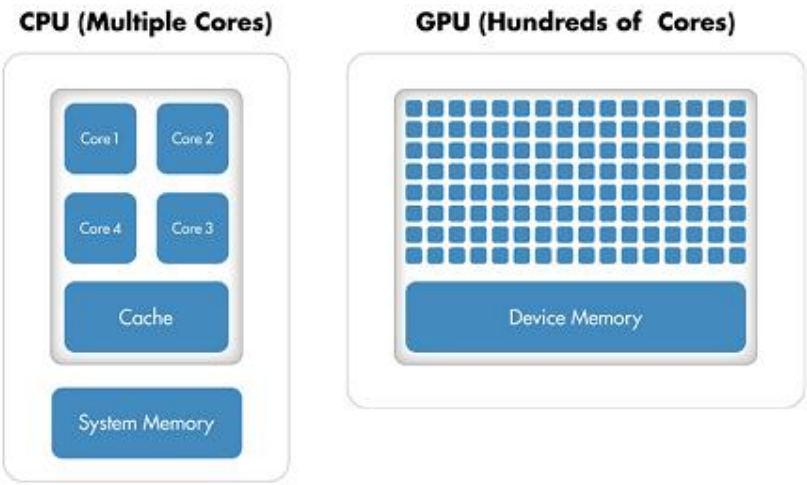
- جنبه داده چندگانه multiple data یعنی باوجود اینکه دستورات بلاک spmd در تمامیlab ها یکسان است، اما هر lab داده های متخلف و یکتا برای آن کد دارد. درنتیجه مجموعه داده چندگانه multiple data setsمی‌توانند توسط multiple labs محاسبه شوند.
- برنامه هایی برای اجرا توسط spmd مناسب هستند که نیاز به اجراهای همزمان یک برنامه روی مجموعه داده چندگانه داشته باشند. درحالیکه نیازمند ارتباط یا همگام سازی بین lab ها نیز وجود داشته باشد. ازجمله می‌توان به موارد زیر اشاره کرد :
  - برنامه هایی که زمان زیادی برای اجرا نیاز دارند. دراین حالت spmd اجازه می‌دهد که تعدادی lab محاسبات مسئله را بصورت همزمان انجام دهند.
  - برنامه هایی که روی مجموعه داده های بزرگ کار می‌کنند. در این حالت spmd اجازه می‌دهد که داده ها بین تعداد lab ها توزیع شود.
- زمانیکه دستورات بلاک spmd روی matlabpool اجرا می‌گردد، تمام خروجی های خط دستور از انجام دهنده ها workers روی پروسس client نمایش داده می‌شود. به این علت که انجام دهنده ها workers همان session های مطلب بدون نمایش هستند و خروجی های گرافیکی (برای مثال دستور figure) نمایش داده نمی‌شوند.
- تابع Composite: اشیا آرایه ای بدون استفاده از دستورات spmd ایجاد می‌کند. این تابع زمانی مفید است که بخواهیم به متغیرهای انجام دهنده ها قبل از دستورات spmd مقدار اولیه بدهیم. همانند مثال زیر (فرض براین است که matlab pool قبلاً اجرا شده و در حالت موازی می‌باشد):

```
c = Composite();          % One element per lab in the pool
for ii = 1:length(c)      % Set the entry for each lab to zero
c{ii} = 0;                % Value stored on each lab
end
```

۴- نحوه استفاده از GPU:

ماشین های چندهسته ای و تکنولوژی hyper-thread دانشمندان، مهندسین و آنالیزگرهای مالی را قادر ساخته است که برنامه های محاسباتی بزرگ را در زمینه های گوناگونی تسریع کنند. امروزه نوع دیگری از سخت افزار واحد پروسس گرافیکی GPU حتی قول انجام محاسبات بیشتری را نیز داده است.

پردازنده گرافیکی GPU که در ابتدا برای افزایش سرعت رندرهای گرافیکی استفاده می‌شده است، امروزه برای محاسبات عملی نیز کاربرد دارد. برخلاف یک CPU معمولی، که تعداد معدودی هسته دارد، یک پردازنده GPU آرایه های موازی متعددی از پروسس های عددی و ممیز شناور و همچنین حافظه سرعت بالا دارد. یک GPU معمولی شامل صدا پروسس کوچک می‌باشد.



اگرچه این تکنولوژی هزینه بر می‌باشد، اما توان خروجی افزایش توسط یک GPU امکان پذیر شده است. اولین نکته اینکه، دسترسی به حافظه یک گلوگاه کندکننده برای محاسبات خواهد بود. داده ها بایستی قبل از محاسبت از CPU به GPU ارسال شده و بعد از انجام عملیات توسط CPU دریافت شود. به این علت که GPU از طریق باس PCI Express به CPU میزبان متصل می‌باشد، دسترسی به حافظه در CPU های معمولی کندتر می‌باشد. در نتیجه بطورکلی افزایش سرعت در محاسبات توسط میزان داده ای که در الگوریتم انتقال می‌یابد، محدود می‌شود. نکته دوم اینکه برنامه نویسی برای GPU در C و یا Fortran نیازمند یک مدل زهنی متفاوت می‌باشد، و بدست آوردن این مهارت زمانبر می‌باشد. علاوه براین برای بهبود کد بایستی زمانی صرف گردد تا برای یک GPU خاص برنامه بهینه گردد.

۴-۱- چگونگی استفاده از GPU در مطلب:

در این قسمت ویژگیهای ابزار محاسبات موازی Parallel Computing Toolbox که شما را قادر می‌سازد کدمطلب خود را با تغییرات اندکی برای GPU قابل اجرا نمایید، معرفی می‌گردد. ابزار Parallel Computing Toolbox در مطلب GPUArray را ارائه می‌نماید. GPUArray یک آرایه خاص با توابع زیادی می‌باشد که اجازه می‌دهد اجرای محاسبات بر روی CUDA-enabled NVIDIA GPU مستقیما از طریق مطلب اجرا شود. این توابع شامل fft، عملگرهای براساس عنصر و تعداد زیادی عملیات های جبری مانند lu و mldivide یا عملگر \ می‌باشد. این جعبه ابزار شما را قادر می‌سازد که از کرنل های CUDA-based GPU مستقیما از طریق مطلب استفاده نمایید.

سوال اساسی این است که آیا اجرا برنامه روی یک GPU باعث افزایش سرعت می‌گردد؟ برای پاسخ به این سوال بایستی دو معیار زیر بررسی گردد :

- بشدت موازی : محاسبات را بایستی بتوان به صدها و یا هزارها واحدهای کاری مستقل تقسیم کرد.
- بشدت محاسباتی : زمان لازم برای محاسبات بایستی بسیار بیشتر از زمان لازم برای انتقال به و از حافظه GPU باشد.

برنامه هایی که این معیارها را نداشته باشند، ممکن است حتی در اجرا روی GPU زمان بیشتری از یک CPU مصرف کنند. توابع اصلی به شرح زیر می‌باشد:

- تابع gpuArray() : این تابع یک آرایه را که در حافظه اصلی قرار دارد، بعنوان پارامتر ورودی دریافت کرده و معادل آن را در روی حافظه GPU می‌سازد.
- تابع gather(): این تابع آرایه موجود در حافظه GPU را دریافت و معادل آنرا در حافظه اصلی می‌سازد.
- سایر توابع که در کتابخانه های paralle.Gpu تعریف شده اند، مستقیما روی متغیرهای حافظه GPU عملیات را بصورت موازی انجام می‌دهند.
- آرایه های خاصی نیز می‌توانند بصورت مستقیم در GPU ایجاد شود بدون اینکه نیاز باشد آنها را از فضای کاری مطلب انتقال دهیم. بعنوان مثال یک ماتریس از صفر می‌تواند مستقیما روی GPU ایجاد شود

```
uxx = parallel.gpu.GPUArray.zeros(N+1,N+1);
```

۴-۲- یک مثال ساده از توابع در مطلب :

```
A = rand(2^16,1);
B = fft (A);
```

برای اجرای عملیات مشابه روی GPU، با استفاده از دستور gpuArray داده ها را از فضای کاری مطلب به حافظه دستگاه انتقال می‌دهیم.

```
A = gpuArray(rand(2^16,1));
B = fft (A);
plot(B);
```

تابع fft از آنجا که پارامتر ورودی آن یک GPUArrayمی‌باشد، مستقیما روی GPU اجرا می‌گردد. می‌توان با استفاده از توابع Gpu-enabled مقادیر یک GPUArray را دستکاری کرده و محاسبات انجام شود.

```
C = gather(B);
```

برای بازگرداندن داده به فضای کاری محلی مطلب از دستور gather استفاده می‌شود. بطوریکه با اجرای این دستور آرایه C یک آرایه double در مطلب می‌باشد.

در این مثال ساده ذخیره زمانی که از اجرای تکی تابع fft بدست می‌آید، اغلب کمتر از زمانی است که برای انتقال یک بردار از فضای کاری مطلب به حافظه دستگاه صرف می‌شود. لازم بذکر است که این زمان ها به سایز آرایه و سخت افزار بستگی مستقیم دارد. هزینه های سربار از انتقال داده می‌تواند عملکرد کلی برنامه را کاهش دهد، بخصوص اگر داده ها مرتباً بین CPU و GPU انتقال یابد درحالیکه عملیات های محاسباتی اندکی در هربار اجرا می‌گردد. حالت بهینه این است که عملیات های محاسباتی زیادی را روی داده در حافظه GPU انجام داده و هر زمان که نیاز باشد، داده ها به CPU انتقال داده شود.

نکته مهم در استفاده از GPU این است که همانند CPU، حافظه آن نیز محدود است. با این تفاوت که برخلاف CPU، پروسس های GPU قابلیت swap کردن حافظه به و از دیسک اصلی را ندارند. در نتیجه قبل از استفاده از GPUArray بایستی اطمینان داشته باشیم که این آرایه از فضای آزاد محدود GPU بیشتر نمی‌شود، بخصوص در زمانیکه می‌خواهیم با آرایه ها بزرگ کار کنیم. دستور gpuDevice یک گزارش از مشخصات کارت گرافیک ازجلمه نام، حافظه کلی و حافظه در درسترس می‌دهد.

۴-۳- برنامه نویسی پیشرفته GPU در مطلب:

ابزار محاسبات موازی یک راه مستقیم برای افزایش سرعت کد مطلب با اجرای آن روی GPU ارائه می‌نماید. این کار به راحتی و فقط با تغییر نوع داده پارامترهای ورودی توابع و استفاده از دستورات مطلب که برای GPUArray ها دوباره بازنویسی شده است، انجام می‌گردد. (یک لبست کامل از این توابع مطلب که GPUArray را پشتیبانی می‌کند در این لینک مشاهده کنید).

برای سرعت بخشیدن به الگوریتم با عملگر های ساده چندتایی روی یک GPU، می‌توان از دستور arrayfun استفاده کرد. این دستور یک تابع را روی هر عنصر از یک آرایه اجرا نمود. به این علت که arrayfun یک تابع GPU-enabled می‌باشد، سرباز زمانی انتقال حافظه فقط برای اولین فراخوانی arrayfun اتفاق می‌افتد و نه در هر بار اجرای تابع.

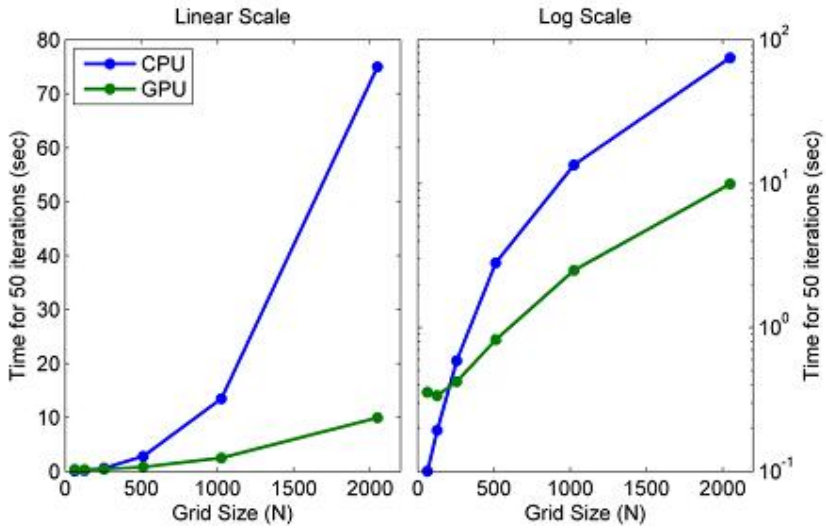
```
y = arrayfun(@foo, x); % Execute on GPU
function y = foo(x)
 y =  1 + x.*(1 + x.*(1 + x.*(1 + ...
   x.*(1 + x.*(1 + x.*(1 + x.*(1 + ...
   x.*(1 + x./9)./8)./7)./6)./5)./4)./3)./2);
```

برنامه نویسانی که کد CUDA را خودشان طراحی کرده اند، می‌توانند از رابط CUDAKernel در ابزار مطلب استفاده کرده و کد را با مطلب یکپارچه کنند. رابط CUDAKernel شما را قادر می‌سازد تا کنترل های بیشتری روی قسمت های خاصی از کد که بصورت گلوگاه می‌باشد، داشته باشید. این رابط یک شی مطلب می‌سازد که به شما دسترسی به کرنل کامپایل شده به کد PTX را می‌دهد (PTX یک مجموعه دستورات اجرای موازی Thread سطح پایین است). و سپس شما می‌توانید با دستور feval کرنل خود را روی GPU با استفاده از آرایه های ورودی و خروجی در مطلب ارزیابی کنید. (برای اطلاع از چگونگی اجرای PTX روی GPU میتوانید به این لینک مراجعه نمایید.)

```
% Setup
kern = parallel.gpu.CUDAKernel('myKern.ptx', cFcnSig)
% Configure
kern.ThreadBlockSize=[512 1];
kern.GridSize=[1024 1024];
% Run
[c, d] = feval(kern, a, b);
```

۴-۴- مقایسه سرعت اجرای GPU و CPU:

برای ارزیابی فواید استفاده از GPU اجرای آنرا برای حل معادلات درجه دوم موجک second-order wave استفاده می‌کنیم. در یک ارزیابی میزان زمان صرف شده برای اجرای ۵۰ بار این الگورتیم روی grid با سایزهای ۶۴، ۱۲۸، ۵۱۲، ۱۰۲۴ و ۲۰۴۸ را روی یک CPU مدل Intel Xeon X5650 و یک پردازنده GPU مدل NVIDIA Tesla C250 اندازه گیری می‌کنیم. برای یک grid با سایز ۲۰۴۸ الگورتیم کاهش ۷.۵ برابری را نشان می‌دهد بطوریکه زمان اجرای آن روی CPU بیش از یک دقیقه می‌باشد، درحالیکه روی GPU کمتر از ۱۰ ثانیه می‌باشد. همانطور که در شکل نشان داده شده است برای Grid با سایزهای کوچک الگورتیم روی CPU سریعتر اجرا می‌شود.



A\b*	Tesla C1060	Tesla C2050 (Fermi)	Quad-core Intel CPU	Ratio (Fermi:CPU)
Single	191	250	48	5:1
Double	63.1	128	25	5:1
Ratio	3:1	2:1	2:1	

FFT	Tesla C1060	Tesla C2050 (Fermi)	Quad-core Intel CPU	Ratio (Fermi:CPU)
Single	50	99	2.29	43:1
Double	22.5	44	1.47	30:1
Ratio	2.2:1	2.2:1	1.5:1	