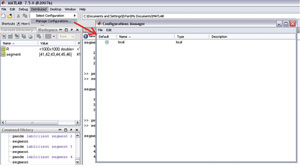
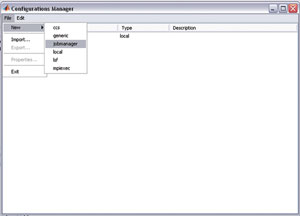
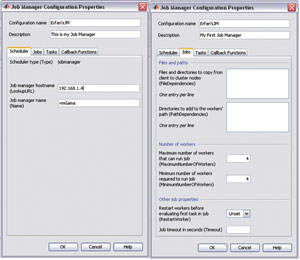
[](http://www.shabakeh-mag.com/Data/Gallery/2011/11/ParallelMatlab4.jpg)

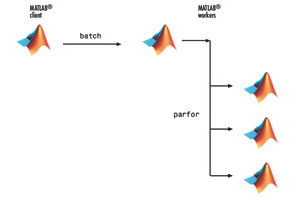
در صورتی که از نرم‌افزار Matlab روی یک كامپيوتر مجهز به پردازنده چند هسته‌ای یا یک كامپيوتر چند پردازنده‌ای استفاده ‌شود، مي‌‌توان از تنظیمات که در سیستم موجود است، استفاده كرد.در این حالت، Job Manager به تعداد هسته‌های سیستم (حداکثر هشت عدد Worker( ایجاد کرده و آن‌ها را آماده انجام محاسبات مي‌كند. در صورت نیاز مي‌‌توان تنظیمات جدیدی در سیستم تعریف كرد. در هر نرم‌افزار Matlab روی هر كامپيوتر مي‌‌توان تعداد زیادی از تنظیمات را ذخیره کرد که یکی از آن‌ها باید به عنوان پیش فرض تعیین شود. به عنوان مثال، مطابق شكل 5 مي‌‌توان یک Job Manager جدید در كامپيوتر تعریف كرد. هر Job Manager تنظیمات اختصاصی خود را دارد که در شكل 6 نمونه‌ای از آن‌ها را مشاهده مي‌كنيد. همان‌طور که در شكل 6 مي‌‌بینید، گزینه‌های مختلفی از جمله نام سرور، حداکثر تعداد Job ها، فایل‌های مورد نیاز همراه برنامه برای ارسال به اعضای کلاستر و خصوصیات سفارشی برای توابع جهت تنظیم در Job Manager وجود دارد.

[](http://www.shabakeh-mag.com/Data/Gallery/2011/11/ParallelMatlab5.jpg)

[](http://www.shabakeh-mag.com/Data/Gallery/2011/11/ParallelMatlab6.jpg)

موازی

در نرم‌افزار Matlab امکاناتی برای استفاده از حلقه‌های For موازی برای اسکریپت‌نویسی وجود دارد. حلقه‌های For موازی حلقه‌هایی هستند که قسمتی از آن‌ها روی ماشین کلاینت Matlab اجرا شده و قسمتی از آن‌ها به صورت موازی به مجموعه ماشین‌های Matlab Workers ارسال مي‌‌شود‌ (شكل 7)داده‌های مورد نیاز ماشین‌های کارگر از کلاینت به آن‌ها ارسال شده و پس از انجام محاسبات، نتایج به کلاینت باز‌گردانده خواهد شد. نکته قابل توجه اين که هر اجرای حلقه For موازی در یک ماشین کارگر یک Iteration بوده و هر کدام از آن‌ها از یکدیگر مستقل بوده و تضمینی برای همگامی اجرای آن‌ها وجود ندارد. بنابراين، تنها حلقه‌هایی که محاسبات و داده‌های آن‌ها امکان اجرا به صورت جداگانه را داشته باشند، با حلقه For موازی قابل پیاده سازی هستند.

[](http://www.shabakeh-mag.com/Data/Gallery/2011/11/ParalellMatlab7.jpg)

**شکل 7**

برای استفاده از حلقه‌های موازی For باید مجموعه موتور پردازش موازی نرم‌افزار را فعال كرد. برای این کار، باید با استفاده از تابع Matlabpool،Scheduler را فعال کرده و سپس تعداد خاصی از ماشین‌های کارگر را رزرو كنيم. با توجه به تنظیمات در Scheduler ماشین‌های کارگر انتخابی مي‌‌توانند ماشین‌های فیزیکی بوده که در یک کلاستر قرار داره یا این‌که مفهومی مجازی روی ماشین کلاینت Matlab باشند.در بخش قبل با سفارشی‌سازی تنظیمات Scheduler آشنا شدی. برای اینکه شروع کنی-Session با ماشین‌های کارگر بر اساس تنظیمات پیش فرض Scheduler، باید عبارت Matlabpool را در خط فرمان نرم‌افزار وارد كنی. این تابع، در صورت وجود تنظیمات مناسب ماشین‌های کارگر راه دور را برای انجام محاسبات موازی آماده کار میکنه.اگه تغییر نداده باشی. برای Scheduler تابع مذکور به ازای هر هسته پردازنده سیستم، یک کارگر روی ماشین محلی ایجاد خواهد میکنه. حداکثر تعداد این کارگرها، هشت عدد خواهد بود.

توجه کن که در صورتی که از تابع Matlabpool برای ایجاد Session استفاده نكنی، حلقه‌های Parfor به صورت سریال اجرا میشن . حال، تابعی مطابق آنچه در فهرست یک اومده تعریف كرده و با عنوان benchmark.m ذخیره می کنی این تابع ساده، یک ماتریس به عنوان ورودی دریافت می کنه و عملیاتی روی اون انجام مي‌‌ده. پس از تعریف تابع نمونه، نوبت به نوشتن اسکریپت تولید متغیرها (ماتریس مرجع) و استفاده از تابع بالا مي‌‌رسه. برای این منظور، اسکریپتی مطابق فهرست 2 ایجاد مي‌كني.در این اسکریپت، ابتدا ماتریسی از اعداد تصادفی با ابعاد 2 به توان 20 و 16 تولید مي‌‌شه. سپس، در صورتی که Matlabpool فعال شد، آن ‌را مي‌‌بندی و تابع مذکور را فراخوانی مي‌كنی. در این حالت، اجرای حلقه Parfor به صورت سریال اتفاق خواهد افتاد. در ادامه Matlabpool با تعداد کارگرهای دلخواه که در بالای اسکریپت مشخص شده‌، فراخوانی شده و تابع دوباره فراخوانی مي‌‌شه. در این حالت، حلقه به صورت موازی به اجرا در خواهد آومد. در انتها، خروجی‌های به دست آمده نمایش داده شده و میزان افزایش سرعت نیز محاسبه شده و نمایش داده مي‌‌شود. پس از اجرای اسکریپت فوق، نتایج به دست آمده مطابق با فهرست 3 خواهد بود.

function [ T ] = benchMark( Ain, E )  
% تعریف متغیرها  
    Msz = size(Ain,1);  
    a = zeros(Msz,1,›single›);  
    R = zeros(Msz,1,›single›);  
    k1 = single(pi/4);  
    k2 = single(pi/5);  
    A = single(Ain);  
% شروع تایمر      
    tstart = tic;  
% انجام محاسبات سنگین      
    parfor (e=1:E)  
    a = A(:,e);  
    R = k2\*a - k1.^2\*a.^2 ...  
        + k2.^2\*a.^3 - k1.^3\*a.^4 ...  
        + k2.^4\*a.^5 - k1.^5\*a.^6 ...  
        + k2.^7\*a.^7 - k1.^8\*a.^8 ...  
        + k2.^9\*a.^9 - k1.^10\*a.^10 ...  
        + k2.^11\*a.^11 - k1.^12\*a.^12 ...  
        + k2.^13\*a.^13 - k1.^14\*a.^14 ...  
        + k2.^15\*a.^15 - k1.^16\*a.^16 ...  
        + k2.^17\*a.^17 - k1.^18\*a.^18 ...  
        + k2.^19\*a.^19 - k1.^20\*a.^20;  
end  
% پایان تایمر  
% برگرداندن زمان محاسبه  
T = toc(tstart);  
end

**فهرست 1- تابع نمونه برای انجام محاسبات سنگین**

clear all;  
 % ایجاد متغیرها  
NLNcols = 2^20;  
E = 16;  
% تعیین تعداد کارگرها  
noWorkers = 2;  
  
%تولید ماتریس رفرنس  
Aref = rand(NLNcols,E);  
  
%% محاسبات با یک ماشین کارگر  
%==========================================================================  
if matlabpool(‹size›) > 0  
    matlabpool close force;  
end  
   
% دریافت ماتریس ورودی  
A = Aref;  
T\_single\_CPU = benchMark( A, E );  
  
%% محاسبات با چند ماشین کارگر  
%==========================================================================  
isOpenCorr = matlabpool(‹size›) == noWorkers;  
if ~isOpenCorr,  
    matlabpool close force  
    matlabpool(noWorkers)  
end  
  
% دریافت ماتریس ورودی  
A = Aref;  
T\_multi\_CPU = benchMark( A, E );  
   
%% PRINT DATA  
fprintf(‹=============================================\n›);  
strCPU1 = ‹# Workers: %d  ->   CPU Time [s]:     %8.3f\n›;  
fprintf(strCPU1, 1, T\_single\_CPU);  
   
strCPU1 = ‹# Workers: %d  ->   CPU Time [s]:     %8.3f\n›;  
fprintf(strCPU1, noWorkers, T\_multi\_CPU);  
  
fprintf(‹=============================================\n›);   
str = ‹Speed-up; 1-CPU / M-CPU [-]:         %8.3f\n›;  
fprintf(str, T\_single\_CPU/T\_multi\_CPU);

**فهرست 2- اسکریپت استفاده کننده از تابع نمونه**

=============================================  
# Workers: 1  ->   CPU Time [s]:       58.992  
# Workers: 2  ->   CPU Time [s]:       30.837  
=============================================  
Speed-up; 1-CPU / M-CPU [-]:            1.913

**فهرست 3- افزایش سرعت محاسبات با استفاده از Par for به میزان 1,92 برابر برای 2 کارگر (هسته)**

 همان طور که مشاهده مي‌‌كنيد، با انجام محاسبات به صورت موازی به افزایش سرعتی مطابق با 92,1 برابر دست یافته‌ایم. مي‌‌توان با افزایش متغیر no Workers به مقادیر بالاتر روی پردازنده‌های قدرتمند‌تر و چند هسته‌ای افزایش سرعت را برای تعداد هسته‌ها و کارگرهای بیشتر نیز محاسبه كرد.