How to use Matlab Parallel Computing Toolbox

Note: The documentation as it pertains to SEAS Compute clusters (such as HPC) is not relevant anymore. However, the actual code might still be relevant.

Matlab Parallel Computing Toolbox (PCT) is now available at SEAS as a part of Matlab r2010a.

We currently support only 'local' parallel mode, i.e running within a single server. The recommended best practice is to run on the hpc cluster interactively or using Matlab scripting. Each compute node supports up to 8-way parallelism and has 16GB of RAM available.

Using Matlab PCT on the hpc cluster

Once on hpc.seas.harvard.edu, request an interactive session from the SGE scheduler

```
qlogin -pe smp 4
```

The above requests 4 'slots' (CPU cores) within a single node.

You can of course also run Matlab .m scripts on the hpc cluster non-interactively. For instructions, please see here.

You can now launch Matlab in non-GUI mode as follows:

```
module load packages/matlab/r2010a
matlab -nodisplay -nosplash
```

Once you get the command prompt, you need to setup the Matlab parallel environment:

```
matlabpool open local 4
```

The above creates a 4-way SMP (shared memory parallel) environment (basically leveraging thread based or OpenMP parallelism under the hood).

Once the environment is set, we can run the following quick benchmark that tests multiply-add (MAD) performance leveraging single CPU core vs leveraging 4 CPU cores and PCT routines:
% this test below will time matrix multiply-add (MAD) on CPU vs using
% Parallel Computing Toolbox
%
%echo off
clear all
sizze=10240;
A =rand([sizze sizze]);
B = inv(A);
format short

% CPU
disp('CPU ::')
tic;
Acpu = A;
Bcpu = B;
% Multiply-add on CPU
Ccpu = Bcpu * Acpu - eye(size(Acpu));
% pull a result
minCcpu = abs(min(min(Ccpu)));
time_cpu=toc;
maxCcpu = abs(max(max(Ccpu)));
S=whos('Acpu');
sizeGB_cpu=getfield(S,'bytes')/1024/1024/1024;
clear Acpu Bcpu Ccpu

% PCT
disp('PCT ::')
tic;
Apct = distributed(A);
Bpct = distributed(B);
Dpct= distributed(eye(size(Apct)));  % Multiply-add is now computed on PCT
Cpct = (Bpct * Apct - Dpct);
% pull a result
minCpct = abs(min(min(Cpct)));
time_pct=toc;
maxCpct = abs(max(max(Cpct)));
clear Apct Bpct Cpct

echo on
disp('Time (s), CPU vs PCT')
[ time_cpu time_pct ]
disp(' Matrix size (GB), on CPU ')
[ sizeGB_cpu ]
disp('Error (arbitrary units)')
[ abs(minCcpu-minCpct) abs(maxCcpu-maxCpct) ]

On one of the compute nodes on the hpc cluster, the CPU execution time is ~35 seconds vs ~118 seconds using PCT. That is, the parallel execution for MAD (as implemented here) is roughly 3.3 times slower.

Matlab PCT offers a myriad of functions, many of which certainly show larger than 1x speed-ups. For example the following 'for loop demo' runs faster using PCT (keeping in mind that one should avoid 'for loops' like the one below at all cost in Matlab in the first place):
tic
istep=1024*100;
for i=1:istep
    A(i) = sin(i*2*pi/istep);
end
toc
clear A

tic
parfor i=1:istep
    A(i) = sin(i*2*pi/istep);
end
toc

This simple 'for loop' example takes ~29 seconds on a single CPU core and ~7 seconds using 4-way PCT execution. That is, the speed up is roughly fourfold. Notably if the problem size is small the PCT execution is slower, e.g. if 'istep=1024', the PCT execution is roughly ten times slower due to the overhead from initializing the 4 'Matlab workers' compared to actual compute effort needed.

More information

For more information, please see the full PCT documentation here.

If you are interested in Matlab Distributed Computing Toolbox (MPI based distributed memory parallel solution), which is currently not available on the hpc cluster, please contact irchelp@seas.harvard.edu.