Introduction to the MATLAB Parallel Toolbox

Marinus Pennings

October 10, 2017
Outline

- Multi threading in MATLAB
- Parallel Pools
- parfor
- spmd
- distributed
- GPU computing
- Cluster Profiles
- MATLAB batch command
- Remote job submission
Short course home page:
https://hprc.tamu.edu/training/matlab_parallel_toolbox.html

Matlab source codes:
- On the course home page
- On ada: /scratch/training/MATLAB-PCT/matlab.zip
- On terra: /scratch/training/MATLAB-PCT/matlab.zip
Multi threading
Multi threading

MATLAB automatically executes a large number of operators multi threaded
- Transparent to user
- Array/Matrix operations
- Elementwise operators

```matlab
>> feature('NumThreads', N);
>> old = maxNumCompThreads(N);
```

HPRC plug

Average Desktop/Laptop has 4 to 8 cores. HPRC cluster terra has 28 cores (20 on ada, some nodes have 40 cores)
Parallel Pool
Parallel Pool

```matlab
>> N=3;
>> p=parpool(N);
```
Parallel Pool

Main MATLAB

>> delete(p)

OR

>> delete(gcp);

Main MATLAB

special variable

<Diagram showing parallel pool and MATLAB commands>
parfor

\[
\text{for } i=1:100 \\
\quad d = d + i; \\
\quad B(i) = R(i) + c \\
\text{end}
\]

\[
\text{parfor } i=1:100 \\
\quad d = d + i; \\
\quad B(i) = R(i) + c \\
\text{end}
\]

(Workers are idle, waiting for more work to do)
1) Main MATLAB sends data and code to workers

```matlab
parfor i=1:100
    d = d + i;
    B(i) = R(i) + c
end
```

(Before sending data, main MATLAB needs to classify all the variables in the loop)
parfor

1) Main MATLAB sends data and code to workers

2) Workers execute assigned iterations

(Main MATLAB waiting for results)
1) Main MATLAB sends data and code to workers
2) Workers execute assigned iterations
3) Workers send results back. Main MATLAB combines results

parfor

\begin{align*}
\text{parfor } & i = 1:100 \\
& d = d + i; \\
& B(i) = R(i) + c \\
\end{align*}

\textbf{reduce } d

\begin{align*}
\text{parfor } & i = 1:25 \\
& d = d + i; \\
& B(i) = R(i) + c \\
\end{align*}

\begin{align*}
\text{parfor } & i = 26:50 \\
& d = d + i; \\
& B(i) = R(i) + c \\
\end{align*}

\begin{align*}
\text{parfor } & i = 51:75 \\
& d = d + i; \\
& B(i) = R(i) + c \\
\end{align*}

\begin{align*}
\text{parfor } & i = 76:100 \\
& d = d + i; \\
& B(i) = R(i) + c \\
\end{align*}
parfor

parfor \( i = 1:100 \)
\[
\begin{align*}
  d &= d + i; \\
  B(i) &= R(i) + c
\end{align*}
\]
end

1) Main MATLAB sends data and code to workers
2) Workers execute assigned iterations
3) Workers send results back. Main MATLAB combines results
4) Main MATLAB gets control back

(continues executing statements after parfor)

(Workers will be idle again, waiting for more work to do)
SPMD block is a construct where all the workers will execute the code in the SPMD block concurrently.

```matlab
spmd
    id = labindex;
    tot = numlabs;
    a = ones(tot)*id;
end
```

(Workers are idle, waiting for more work to do)
SPMD

1) Main MATLAB sends code block to workers

```matlab
spmd
    id = labindex;
    tot = numlabs;
    a = ones(tot) * id
end
a1 = a{1}
```

- id = 1; tot = 4; a = ones(4)*1;
- id = 2; tot = 4; a = ones(4)*2;
- id = 3; tot = 4; a = ones(4)*3;
- id = 4; tot = 4; a = ones(4)*4;
SPMD

1) Main MATLAB sends code block to workers
2) Workers execute code in SPMD block

```
spmd
    id = labindex;
    tot = numlabs;
    a=ones(tot)*id
end

a1 = a{1}
```

(Main MATLAB waiting)

```
id = 1;
tot = 4;
a=ones(4)*1;
```

```
id = 2;
tot = 4;
a=ones(4)*2;
```

```
id = 3;
tot = 4;
a=ones(4)*3;
```

```
id = 4;
tot = 4;
a=ones(4)*4;
```
SPMD

1) Main MATLAB sends code block to workers
2) Workers execute code in SPMD block
3) Main MATLAB gets control back

```matlab
spmd
    id = labindex;
    tot = numlabs;
    a=ones(tot)*id
end
a1 = a{1}
```
SPMD Communication

- `labSend(var, id)` → sends variable “var” to worker “id”
- `var=labReceive(id)` → receives data from “id” and assigns to “var”
- `vf=labSendReceive(it, if, vt)` → sends “vt” to “it”, receive data from “if” and assign to “vf”

spmd
  id=labindex; n=numlabs;
  if (id == n)
    labSend(1,id);
  else if (id == 1)
    id=labReceive(n);
  end

(Main MATLAB waiting)

Sending id=4

Id = labReceive(4);
%empty
% empty
labSend(4,1);
Distributed Arrays

Conceptually very similar to a regular array.

- Many regular matrix operators available for distributed arrays.
- Elements can be of any type
- Elements distributed over the workers
- **Matlab will automatically use parallel version of operator** if operand is distributed variable

```matlab
>> methods('distributed');
```

```
workspace
workspace
workspace
workspace
workspace
```
Distributed Arrays

Conceptually very similar to a regular array.

- Many regular matrix operators available for distributed arrays.
- Elements can be of any type
- Elements distributed over the workers
- **Matlab will automatically use parallel version of operator** if operand is distributed variable

```
>> b=rand(1000);
>> a=distributed(b);
```

```
a(1:250)
workspace
```
```
workspace
```
```
workspace
```
```
workspace
```
```
workspace
```
GPU Programming

What is a GPU?

- Accelerator card
- Thousands of small computing cores
- Dedicated high speed memory.
GPU Programming

- MATLAB provides GPU versions for a large number of MATLAB operators.
- Completely transparent to user
- **Matlab will automatically use GPU version of operator If operand is GPU variable**

```matlab
>> methods('gpuArray');
```
Copy to/from GPU

- **function:** \( \text{var2} = \text{gpuArray}(\text{var1}) \) → copy variable “v1” on host to GPU and name “v2”
- **function:** \( \text{var2} = \text{gather}(\text{var1}) \) → copy variable “v1” on GPU to host and assign to variable “var2”
- Use convenient functions to create data directly on the GPU

```
>> a = zeros(100);
>> ag = gpuArray(a);
>> bg = gpuArray.rand(100);
```
parpools revisited

What if you want more workers than cores/nodes?

Remember from creating parpool, didn’t provide cluster profile, so using default ‘local’ profile!
parpools revisited

What if you want more workers than cores/nodes?

Remember from creating parpool, didn’t provide cluster profile, so using default ‘local’ profile!

or want to distribute workers over multiple nodes?
(e.g. so each worker can use more threads)
parpools revisited

What if you want more workers than cores/nodes?

Local Profile:
- Workers must run on same computer/node as main MATLAB
- Limited to the number of cores on a computer/node → 28 workers on terra (20 on ada)
- Part of the MATLAB Parallel Toolbox

or want to distribute workers over multiple nodes? (e.g. so each worker can use more threads)
Cluster Profile

- Number of workers only limited by license (currently 96, shared)
- Integrates with Batch scheduler (e.g. slurm and lsf)
  - Will actually submit lsf/slurm jobs
  - Workers will be running on the compute nodes

MDCS license
Importing Cluster Profile

Only on HPRC

```matlab
>> profile = parallel.importProfile(<PATH>);
```

- Only need to import cluster profile once
- Pre-created profile located in $MATLABDIR/profiles/TAMU

**tamu convenience function:**

```matlab
>> tamu_import_TAMU_clusterprofile();
```

- Wrapper around parallel.clusterProfile()
- No need to provide location of pre-created profile
- Creates directory in scratch directory to store meta data
Cluster Properties

How to attach properties (e.g. workers/threads/time)?

Defining cluster properties:

```matlab
>> help TAMUClusterProperties % to see all options
>> tp = TAMUClusterProperties();
>> tp.workers(4);
>> tp.walltime('02:00);
```

Attaching properties to cluster profile:

```matlab
% attach the properties to the profile
>> profile = tp.tamu_set_profile_properties(tp);
```
MATLAB batch function

Offloads a script or function to worker(s), control is returned immediately, Job object is returned

```matlab
>> j=batch(cluster obj>, 'myscript', 'Pool', N); % offloads script (start pool)
>> j=batch(<cluster obj>, @myfunc, N, {x1, x2}); % offloads function
```

Retrieving Job info and results:

```matlab
>> r=j.State(); %
>> j.wait(); % offloads script (start pool)
>> j.load(); % offloads script (start pool)
>> res=j.fetchOutputs(); % offloads function
```
Remote batch submission

Submit jobs from user’s local MATLAB session (laptop/desktop)

from Matlab command line

```matlab
>> tp = TAMUClusterProperties();
>> tp.hostname('terra.tamu.edu');
>> tp.user('<netid>);
>> j1=tamu_run_batch(tp,'mytest');

% or run a function

>> cp=tamu_set_profile_properties(tp);
>> j2=batch(cp,@myfun,1,{a,b});
```

using the MATLAB app

Download framework and app at https://hprc.tamu.edu/files/HPRC.mlappinstall
https://hprc.tamu.edu/wiki/SW:Matlab#Running_(parallel)_Matlab_Scripts_on_HPRC_compute_nodes
Questions?

For additional information/help:
See Wiki:  https://hprc.tamu.edu/wiki/SW:Matlab
Send email: help@hprc.tamu.edu