Workshop: Parallel Computing with MATLAB and Scaling to HPCC

Raymond Norris
MathWorks
Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting
What’s Not Being Covered Today?

- Data Parallel
- MapReduce
- MPI
- Simulink
Let’s Define Some Terms

cli·ent noun ˈkli-ənt
1 : MATLAB session that submits the job

com·mu·ni·cate job adjective ˈkə-ˈmyü-ˈnət-ˌkāt ˈjāb
1 : a job composed of tasks that communicate with each other, running at the same time

in·de·pen·dent job adjective ˈin-di-ˈpen-dənt ˈjāb
1 : a job composed of independent tasks, with no communication, which do not need to run at the same time

lab noun ˈlab
1 : see worker
...a Few More Terms

MATLAB pool noun \mat-lab\ \'pül\'
1 : a collection of workers

MDCS abbreviation
1 : MATLAB Distributed Computing Server

SPMD abbreviation
1 : Single Program Multiple Data

worker noun \'wər-kər\n1 : headless MATLAB session that performs tasks
MATLAB Parallel Computing Solution

Desktop Computer

MATLAB Desktop (Client)

Local

Parallel Computing Toolbox

Computer Cluster

MATLAB Distributed Computing Server

Cluster

Scheduler
Typical Parallel Applications

- Massive for loops (parfor)
  - Parameter sweep
    - Many iterations
    - Long iterations
  - Monte-Carlo simulations
  - Test suites
- One-Off Batch Jobs
- Partition Large Data Sets (spmd)

Task Parallel Applications

Data Parallel Applications
Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting
But Before We Get Started…

- Do you preallocate your matrices?
Effect of Not Preallocating Memory

```matlab
>> x = 4;
>> x(2) = 7;
>> x(3) = 12;
```

Resizing Arrays is Expensive
Benefit of Preallocation

\[
\begin{align*}
\texttt{>> } x & = \text{zeros}(3,1); \\
\texttt{>> } x(1) & = 4; \\
\texttt{>> } x(2) & = 7; \\
\texttt{>> } x(3) & = 12;
\end{align*}
\]
Let’s Try It…

```matlab
>> len = 10e7;
>> tic, for idx = 1:len, A(idx) = idx; end, toc
Elapsed time is 41.678428 seconds.
>>
>> B = ones(1,len);
>> tic, for idx = 1:len, B(idx) = idx; end, toc
Elapsed time is 8.063080 seconds.
```
Getting Started With the MATLAB Pool
The MATLAB Pool

MATLAB Desktop (Client)

Worker

Worker

Worker

Worker

Worker

Worker
Connecting to HPCC to Run MATLAB

ssh -X USERNAME@hpc-login1.usc.edu

## For bash users
% cp ~matlab/setup_matlab.sh ~/
% source setup_matlab.sh

## For tcsh users
% cp ~matlab/setup_matlab.csh ~/
% source setup_matlab.csh

% matlab_local ## or matlab_cluster

ssh -X COMPUTE-NODE
. /usr/usc/matlab/2013a/setup.[c]sh
% matlab &
Starting a MATLAB Pool...

Bring up the Windows Task Manager or Linux `top`

Start MATLAB

Open a MATLAB pool with two workers using the local profile

```
>> matlabpool(2)
Starting matlabpool using the 'local' profile ... connected to 2 workers.
```

Maximum of 12 local workers
One MATLAB Pool at a Time

Even if you have not exceeded the maximum number of workers, you can only open one MATLAB pool at a time.
Stopping a MATLAB Pool

>> matlabpool close
Sending a stop signal to all the workers ... stopped.

f(x) >>
Add Shortcut for Starting the MATLAB Pool

```
if matlabpool('size') == 0
    matlabpool open local 2
end
```
Add Shortcut for Stopping the MATLAB Pool

```
if matlabpool('size')>0
    matlabpool close
end
```
Toolbox Support for Parallel Computing

```matlab
options = optimset;
options = optimset(options, 'Display', 'none');
options = optimset(options, 'TolFun', param.tol);
options = optimset(options, 'OutputFcn', @(x,it,f) helper.plotFcn(x,it,f,dimensions), @helper.myOut);
options = optimset(options, 'Algorithm', 'active-set');

options = optimset(options, 'UseParallel', 'always');

startTic = tic;
[tmp, tmp, tmp, output] = ...
    fmincon(@(x) helper.objFcn(x, dimensions.R), x0, [], [], [], [], [], lb, ub, [], options); %#ok<ASGLU>
toc(startTic);

helper.plotOptimSummary(output, dimensions);
```
Products That Support PCT

- Bioinformatics Toolbox
- Communications System Toolbox
- Embedded Coder
- Global Optimization Toolbox
- Image Processing Toolbox
- Model-Based Calibration Toolbox
- Neural Network Toolbox
- Optimization Toolbox
- Phased Array System Toolbox
- Robust Control Toolbox
- Signal Processing Toolbox
- Simulink
- Simulink Coder
- Simulink Control Design
- Simulink Design Optimization
- Statistics Toolbox
- SystemTest

parfor: The Parallel for Loop
Using the **parfor** Construct

- In order to convert a `for` loop to a `parfor` loop, the `for` loop must at least be:
  - Task independent
  - Order independent
Order Independent?

```matlab
>> parfor idx = 1:4, idx, end
ans =
    4
ans =
    3
ans =
    2
ans =
    1
```
What If a MATLAB Pool Is Running?

```matlab
>> matlabpool(2)
Starting matlabpool using the 'local' profile ... connected to 2 workers.
>> parfor idx = 1:4, idx, end
ans =
  3
ans =
  2
ans =
  1
ans =
  4
```
The Mechanics of `parfor` Blocks

c = pi;
a = zeros(10, 1)
for idx = 1:10
    a(idx) = idx * c;
end
a

Pool of MATLAB Workers
The Mechanics of `parfor` Blocks

```matlab
% Auto-load balancing
% Pool of MATLAB Workers

c = pi;
a = zeros(10, 1);
parfor idx = 1:10
    a(idx) = idx * c;
end
a
```
Example: Hello, World!

1. Code the example below. Save it as `foreexample.m`

```matlab
function a = foreexample()
    tic
    for idx = 1:100
        a(idx) = myfcn(idx);
        pause(0.25)
    end
    toc
```

```bash
>> foreexample
```
Example: Hello, World! (2)

2. Code the helper function. Save it as `myfcn.m`. Time and run it.
Example: Hello, World! (3)

3. Parallelize the `for` loop and save it as `parforexample.m`

4. Start a MATLAB pool and run it. Change the size of the Pool. What speed ups do you get?

```matlab
function a = parforexample()
    tic
    parfor idx = 1:100
        a(idx) = myfcn(idx);
        pause(0.25)
    end
    toc
```

```
>> parforexample
```
Example: Break It (1)

5. Add a dependency to the `parfor` loop. Look at the code analyzer messages.

```matlab
def function a = parforbug()
    tic
    parfor idx = 2:100
        a(idx) = myfcn(idx) + a(idx-1);
        pause(0.25)
    end
    toc
end
```

```bash
>> parforbug
```
The variable $a$ cannot be properly classified
Constraints

- The loop variable cannot be used to index with other variables
- No inter-process communication. Therefore, a `parfor` loop cannot contain:
  - break and return statements
  - global and persistent variables
  - nested functions
  - changes to handle classes
- Transparency
  - Cannot “introduce” variables (e.g. `eval`, `load`, `global`, etc.)
  - Unambiguous Variables Names
- No nested `parfor` loops or `spmd` statement
This is Great! Should I Get Linear Improvement?

- Not exactly
  - Too little work, too much data
- Are you calling BLAS or LAPACK routines?
- What are you timing?
  - MATLAB Profiler
- Amdahl’s Law
  - \( SU(N) = \frac{1}{(1-P) + \frac{P}{N}} \)
Optimizing a `parfor` Loop

- Should I pre-allocate a matrix?
  - There is no significant speedup, if any, in pre-allocating the matrix

- Should I pre-assign large matrices before the `parfor`?
  - Yes, if they’re going to be referenced after the `for` loop (to be explained why later)
  - Otherwise, do all the large creation on the workers
  - So if I have a `for` loop with 100 iterations and 10 workers, are each of the matrices create 10 times? Or 100 times?
    - 100 times. See later for minimizing this.
**parfor Variable Classification**

- All variables referenced at the top level of the `parfor` must be resolved and classified

<table>
<thead>
<tr>
<th>Classification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop</td>
<td>Serves as a loop index for arrays</td>
</tr>
<tr>
<td>Sliced</td>
<td>An array whose segments are operated on by different iterations of the loop</td>
</tr>
<tr>
<td>Broadcast</td>
<td>A variable defined before the loop whose value is used inside the loop, but never assigned inside the loop</td>
</tr>
<tr>
<td>Reduction</td>
<td>Accumulates a value across iterations of the loop, regardless of iteration order</td>
</tr>
<tr>
<td>Temporary</td>
<td>Variable created inside the loop, but unlike sliced or reduction variables, not available outside the loop</td>
</tr>
</tbody>
</table>

> `web([docroot '/distcomp/advanced-topics.html#bq_of7_-1'])`
Variable Classification Example

```
a = 0;
c = pi;
z = 0;
r = rand(1,10);
parfor idx = 1:10
    a = idx;
    z = z+idx;
    b(idx) = r(idx);
    if idx<=c
        d = 2*a;
    end
end
```

1. Loop
2. Temporary
3. Reduction
4. Sliced Output
5. Sliced Input
6. Broadcast
After the `for` loop, what is the type and the value of each variable?

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>broadcast</td>
<td>ones(1:10)</td>
</tr>
<tr>
<td>b</td>
<td>temp</td>
<td>undefined</td>
</tr>
<tr>
<td>c</td>
<td>temp</td>
<td>undefined</td>
</tr>
<tr>
<td>d</td>
<td>sliced</td>
<td>1:10</td>
</tr>
<tr>
<td>e</td>
<td>reduction</td>
<td>55</td>
</tr>
<tr>
<td>f</td>
<td>temp</td>
<td>5</td>
</tr>
<tr>
<td>g</td>
<td>reduction</td>
<td>20</td>
</tr>
<tr>
<td>h</td>
<td>temp</td>
<td>10</td>
</tr>
<tr>
<td>j</td>
<td>temp</td>
<td>0.0000 + 1.0000i</td>
</tr>
<tr>
<td>s</td>
<td>broadcast</td>
<td>rand(1,10)</td>
</tr>
<tr>
<td>idx</td>
<td>loop</td>
<td>undefined</td>
</tr>
</tbody>
</table>

`>> what_is_it_parfor`
Sliced Variables

- An indexed variables, parceled out to each worker
  - Indexing at the first level only and for () or {} 
  - Within the list of indices for a sliced variable, one of these indices is of the form $i$, $i+k$, $i-k$, $k+i$, or $k-i$, where $i$ is the loop variable and $k$ is a constant or a simple (non-indexed) broadcast variable; and every other index is a constant, a simple broadcast variable, colon, or end

<table>
<thead>
<tr>
<th>Not Valid</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(i+f(k),j,:,3)</td>
<td>A(i+k,j,:,3)</td>
</tr>
<tr>
<td>A(i,20:30,end)</td>
<td>A(i,:,end)</td>
</tr>
<tr>
<td>A(i,:,s.field1)</td>
<td>A(i,:,k)</td>
</tr>
</tbody>
</table>
Implications of Sliced Variables

What is the value of $A$?

```matlab
>> bad_sliced_matrix
```
Implications of Broadcast Variables

The entire data set $r$ is broadcast to each worker

```
>> broadcast_matrix
```

```matlab
1 - g = zeros(1,10);
2 - r = magic(10);
3 - parfor idx = 1:10
4 - g(idx) = rand*max(eig(r));
5 - end
```
Implications of Broadcast Variables

Could you create \( r \) on the workers instead?

\[
g = \text{zeros}(1,10); \\
parfor \ idx = 1:10 \\
\hspace{1cm} r = \text{magic}(10); \\
\hspace{1cm} g(idx) = \text{rand} \times \text{max} \left( \text{eig}(r) \right); \\
end
\]

\[
>> \text{temporary\_matrix}
\]
Implications of Broadcast Variables

```matlab
>> tic
r = magic(1500); % ~17 MB
g = zeros(1,16);
parfor idx = 1:16
    g(idx) = rand*max(eig(r));
end
toc
Elapsed time is 18.914094 seconds.

>> tic
g = zeros(1,16);
parfor idx = 1:16
    r = magic(1500); % ~17 MB
g(idx) = rand*max(eig(r));
end
toc
Elapsed time is 5.733446 seconds.
```
Implications of Reductions Variables

- Variable appears on both sides of assignment
- Same operation must be performed on variable for all iterations
- Reduction function must be associative and commutative
Implications of Reduction Variables

```
>> x = 0;
parfor idx = 1:10
    x = x+idx;
end
x

>> x2 = [];
parfor idx = 1:10
    x2 = [x2 idx];
end
x2

>> x3 = 0;
parfor idx = 1:32
    if idx<16
        x3 = x3*idx;
    else
        x3 = x3+idx;
    end
end
x3

Error: Different reduction functions are used for the same variable x3.
See Parallel for Loops in MATLAB, "Basic Rules for Reduction Variables".
```
Implications of Temporary Variables

What is the value of $A$? $d$? $idx$?
Variable Assignments Are Not Displayed When Running a `parfor`
MATLAB has a repeatable sequence of random numbers.

When workers are started up, rather than using this same sequence of random numbers, the `labindex` is used to seed the RNG.
**rand in parfor Loops (2)**

```
>> rand('twister',5489)
>> for idx = 1:8, rand, end
ans =
    0.8147
ans =
    0.9058
ans =
    0.1270
ans =
    0.9134
ans =
    0.6324
ans =
    0.0975
ans =
    0.2785
ans =
    0.5469
```
Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting
What If My `parfor` Has a `parfor` In It?

- MATLAB runs a static analyzer on the **immediate** `parfor` and will error out nested `parfor` loops. However, functions called from within the `parfor` that include `parfor` loops are treated as regular `for` loops.

```matlab
>> nestedparfor_bug
>> nestedparfor_fix
```
What’s Wrong With This Code?

Why can we index into C with jidx, but not B?

>> whats_wrong_with_this_code
parfor issue: Indexing With Different Expressions

How can we avoid indexing into \( x \) two different ways?

```matlab
x = rand(10,2);
parfor jidx = 1:10;
    x(jidx,1) = 2;
end
z = x(jidx,2);
```

>> valid_indexing_bug
Create a temporary variable, `x_2nd_col`, to store the column vector. Then loop into the vector using the looping index, `jidx`, rather than the into a matrix.

Note: This doesn’t scale very well if we needed to index into `x` many ways.

```
>> valid_indexing_fix
```

```plaintext
x = rand(10,2);
x_2nd_col = x(:,2);

parfor jidx = 1:10;
    x(jidx,1) = 2;
    z = x_2nd_col(jidx);
end
```
parfor issue: Inadvertently Creating Temporary Variables

What is the code analyzer message? And how can we solve this problem?
Why does the code analyzer think highest is a temporary variable?

>> inadvertent_temporary_bug

```
1 - highest = -inf;
2 - parfor idx = 1:10
3 - value = idx;
4 - if value>highest
5 - highest = value;
6 - end
7 - end
```
parfor issue: Solution

Assign `highest` to the result of a reduction function

```plaintext
parfor idx = 1:10
    value = idx;
    highest = max(highest, value);
end
```
parfor issue: Inadvertently Creating Broadcast Variables

What is the code analyzer message?
Why isn’t \( c \) a sliced variable? What kind is it?
How can we make it sliced?
If we didn’t have the \( b \) assignment, would \( c \) be sliced?

>> inadvertent_broadcast_bug
parfor issue: Solution

Create the additional variables $x$ and $y$, which are sliced

```matlab
>> inadvertent_broadcast_fix
```

```
c = rand(10);
x = c(:,1);
y = c(:,2);
parfor i = 1:10
    a(i) = x(i);
b(i) = y(i);
end
```
Persistent Storage (1)

- I cannot convert the outer loop into `parfor` because it’s in someone else’s top level function. However, if I convert the inner loop into `parfor` in the straightforward manner, we end up sending large data to the workers N times.
Persistent Storage (2)

```matlab
largeData = rand(1e6);
numIters = 1e9;
for outer = 1:100
    parfor inner = 1:numIters
        y(inner) = foo(largeData, inner);
    end
end
```
Solution: Persistent Storage

Store the value in a persistent variable in a function

```matlab
function x = myStorage(x)
    persistent STORED_VALUE
    if nargin > 0
        STORED_VALUE = x;
    end
    x = STORED_VALUE;
end
```
Best Practices for Converting for to parfor

- Use code analyzer to diagnose parfor issues
- If your for loop cannot be converted to a parfor, consider wrapping a subset of the body to a function
- If you modify your parfor loop, switch back to a for loop for regression testing
- Read the section on classification of variables

>> docsearch ‘Classification of Variables’
Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting
What is a Graphics Processing Unit (GPU)

- Originally for graphics acceleration, now also used for scientific calculations
- Massively parallel array of integer and floating point processors
  - Typically hundreds of processors per card
  - GPU cores complement CPU cores
- Dedicated high-speed memory
  - blogs.mathworks.com/loren/2013/06/24/running-monte-carlo-simulations-on-multiple-gpus

* Parallel Computing Toolbox requires NVIDIA GPUs with Compute Capability 1.3 or higher, including NVIDIA Tesla 20-series products. See a complete listing at www.nvidia.com/object/cuda_gpus.html
Performance Gain with More Hardware

Using More Cores (CPUs)

Using GPUs

GPU cores

Device Memory
Programming Parallel Applications (GPU)

- Built-in support with Toolboxes
Programming Parallel Applications (GPU)

- Built-in support with Toolboxes
- Simple programming constructs: `gpuArray`, `gather`
Example: Solving 2D Wave Equation

GPU Computing

- Solve 2\textsuperscript{nd} order wave equation using spectral methods:
  \[
  \frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}
  \]

- Run both on CPU and GPU

- Using \texttt{gpuArray} and overloaded functions

www.mathworks.com/help/distcomp/using-gpuarray.html#bsloua3-1
Benchmark: Solving 2D Wave Equation
CPU vs GPU

Intel Xeon Processor W3690 (3.47GHz),
NVIDIA Tesla K20 GPU
Programming Parallel Applications (GPU)

- Built-in support with Toolboxes
- Simple programming constructs:
  - `gpuArray`, `gather`
- Advanced programming constructs:
  - `arrayfun`, `bsxfun`, `spmd`
- Interface for experts:
  - `CUDAKernel`, MEX support

www.mathworks.com/help/releases/R2013a/distcomp/executing-cuda-or-ptx-code-on-the-gpu.html
GPU Performance – not all cards are equal

- Tesla-based cards will provide best performance
- Realistically, expect 4x to 15x speedup (Tesla) vs CPU
- See GPUBench on MATLAB Central for examples

www.mathworks.com/matlabcentral/fileexchange/34080-gpubench
Criteria for Good Problems to Run on a GPU

- **Massively parallel:**
  - Calculations can be broken into hundreds or thousands of independent units of work
  - Problem size takes advantage of many GPU cores

- **Computationally intensive:**
  - Computation time significantly exceeds CPU/GPU data transfer time

- **Algorithm consists of supported functions:**
  - Growing list of Toolboxes with built-in support
  - Subset of core MATLAB for `gpuArray`, `arrayfun`, `bsxfun`
    - [www.mathworks.com/help/distcomp/using-gpuarray.html#bsloua3-1](http://www.mathworks.com/help/distcomp/using-gpuarray.html#bsloua3-1)
    - [www.mathworks.com/help/distcomp/execute-matlab-code-elementwise-on-a-gpu.html#bsnx7h8-1](http://www.mathworks.com/help/distcomp/execute-matlab-code-elementwise-on-a-gpu.html#bsnx7h8-1)
Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting
Migrating from Local to Cluster

MATLAB client

MATLAB workers

parfor

batch

MATLAB workers
Offload Computations with \texttt{batch}

MATLAB Desktop (Client)

Worker

Worker

Worker

Work

Result
Can’t I Just Use `matlabpool` to Connect to the Cluster/Cloud?

- **MATLAB pool**
  - So long as the compute nodes can reach back to your local desktop, then yes, you can run jobs on the cluster using `matlabpool`.
  - Recall, the MATLAB Client is blocked.
  - Cannot run other parallel jobs.
  - Consumes MDCS licenses while the pool is open, even if they aren’t being used.

- **Batch**
  - Ideal if:
    - the local desktop is not reachable from the cluster, or
    - if I want shutdown my desktop, or
    - if I want submit multiple jobs at once.
Why Can’t I Open a MATLAB Pool to the Cluster?

>> matlabpool(32)

Can’t resolve hostname

Can it resolve the IP address?

>> pctconfig('hostname','12.34.56.78')
Profiles

- Think of cluster profiles like printer queue configurations

Managing profiles
- Typically created by Sys Admins
- Label profiles based on the version of MATLAB
  - E.g. hpcc_local_r2013a

Import profiles generated by the Sys Admin
- Don’t modify them with two exceptions
  - Specify the JobStorageLocation
  - Setting the ClusterSize

Validate profiles
- Ensure new profile is properly working
- Helpful when debugging failed jobs
Import and Validating a Profile

Cluster Profile Manager

Properties

Overall Status: *** Not run

<table>
<thead>
<tr>
<th>Stage</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster connection test (parcluster)</td>
<td>*** Not run</td>
<td></td>
</tr>
<tr>
<td>Job test (createJob)</td>
<td>*** Not run</td>
<td></td>
</tr>
<tr>
<td>SPMD job test (createCommunicatingJob)</td>
<td>*** Not run</td>
<td></td>
</tr>
<tr>
<td>Pool job test (createCommunicatingJob)</td>
<td>*** Not run</td>
<td></td>
</tr>
<tr>
<td>MATLAB pool test (matlabpool)</td>
<td>*** Not run</td>
<td></td>
</tr>
</tbody>
</table>
Submitting Scripts with \texttt{batch}

```matlab
N = 10;
a = zeros(N,2);
disp('Problems with two parameters needed for analysis');
for j = 1:N
    a(j) = rand(1,2);
end
disp(a(1:2,1))
```

```matlab
job = batch('run_sims');
job.wait
job.load
a(1:2,1)
ans =
    1
    2
```
Submitting Functions with `batch`

```matlab
function [a, t] = run_fcn_sims(N)
    a = zeros(N,1);
    disp('Start sims')
    start_t = tic();
    parfor jidx = 1:N
        pause(0.1)
        a(jidx,1) = jidx;
    end
    t = toc(start_t);
    disp('Sims finished')
end
```

```matlab
>> job1 = batch(@run_fcn_sims,2,[100],'matlabpool',3);
>> job2 = batch(@run_fcn_sims,2,[100],'matlabpool',3);
>> job1.wait
>> job1.fetchOutputs(2)
ans =
    4.1782

>> run_fcn_sims
```
Fixing the `batch` Warning Message

Warning: Unable to change to requested working directory. Reason: Cannot CD to C:\Work (Name is nonexistent or not a directory).

- **Call `batch` with `CurrentFolder` set to `.´**
- `job = batch(....,'CurrentFolder','.');`
How Can I Find Yesterday’s Job?

Job Monitor
Final Exam: What Final Exam?

- Choose one of the following:
  - Submit a job that determines the MATLAB directory your task ran in
  - Submit a job that determines the machine that ran your task
    - Hint: system(), hostname.exe

- Clear your MATLAB workspace and get a handle to the job you ran above
Final Exam: Solution (1)

```matlab
>> j = batch(@pwd,1,{});
>> j.wait
>> j.fetchOutputs{1}
ans =
    C:\Work
    _ _
```

```matlab
>>
```
>> j = batch(@system,2,{'hostname'});
>> j.wait
>> j.fetchOutputs{:}
   ans =
       0
   ans =
   AH-RAYN

>>
Recommendations

- Profile your code to search for bottlenecks
- Make use of **code analyzer** when coding `parfor` and `spmd`
- Display the correct amount of verbosity for debugging purposes
- Implement an error handler, including capture of calls to 3rd party functions – **don’t assume calls to libraries succeed**
- Beware of multiple processes writing to the same file
- Avoid the use of global variables
- **Avoid hard coding path and filenames** that don’t exist on the cluster
- Migrate from scripts to functions
- Consider whether or not you’ll need to recompile your MEX-files
- After migrating from `for` to `parfor`, switch back to `for` to make sure nothing has broken
- If calling `rand` in a for loop, while **debugging** call `rand(‘seed’,0)`, to get consistent results each time
- When calling `matlabpool/batch`, parameterize your code
Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting
Troubleshooting and Debugging

- Object data size limitations
  - Single transfers of data between client and workers

<table>
<thead>
<tr>
<th>System Architecture</th>
<th>Maximum Data Size Per Transfer (approx.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-bit</td>
<td>2.0 GB</td>
</tr>
<tr>
<td>32-bit</td>
<td>600 MB</td>
</tr>
</tbody>
</table>

- Tasks or jobs remain in Queued state even though cluster scheduler states it’s finished
  - Most likely MDCS failed to startup

- No results or job failed
  - job.load or job.fetchOutputArguments{:}
  - job.Parent.getDebugLog(job)
System Support
System Requirements

- Maximum 1 MATLAB worker / CPU core
- Minimum 1 GB RAM / MATLAB worker
- Minimum 5 GB of disk space for temporary data directories

GPU
- CUDA-enabled NVIDIA GPU w/ compute capability 1.3 or above [http://www.nvidia.com/content/cuda/cuda-gpus.html](http://www.nvidia.com/content/cuda/cuda-gpus.html)
What’s New In R2013a?

- GPU-enabled functions in Image Processing Toolbox and Phased Array System Toolbox
- More MATLAB functions enabled for use with GPUs, including `interp1` and `ismember`
- Enhancements to MATLAB functions enabled for GPUs, including `arrayfun`, `svd`, and `mldivide (\)`
- Ability to launch CUDA code and manipulate data contained in GPU arrays from MEX-functions
- **Automatic detection and transfer of files required for execution in both batch and interactive workflows**
- More MATLAB functions enabled for distributed arrays
Training: Parallel Computing with MATLAB

- Two-day course introducing tools and techniques for distributing code and writing parallel algorithms in MATLAB. The course shows how to increase both the speed and the scale of existing code using PCT.
  - Working with a MATLAB pool
  - Speeding up computations
  - Task-parallel programming
  - Working with large data sets
  - Data-parallel programming
  - Increasing scale with multiple systems
  - **Prerequisites:** *MATLAB Fundamentals*

- [mathworks.com/training](http://mathworks.com/training)