Parallel Computing in MATLAB @ RCC

February 3, 2026

Alex Townsend, Ph.D.
FSU Research Computing Center

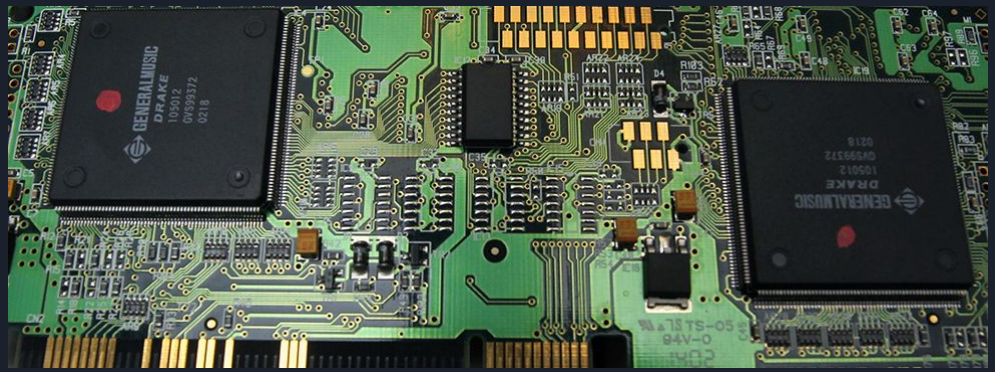
Outline

- Introduction to Parallel Computing
- Parallel Computing Setup
- Automatic Parallelization
- Parallel Loops

Introduction to Parallel Computing

Introduction to Parallel Computing

- Multicore and Distributed Computing
- Shared Memory, Distributed Memory
- Terminology: Threads, Processes
- Caveats and Pitfalls: Loop Dependencies



Basic Concepts in Parallel Computing

- Serial computing is done like reading directions in a user manual.
  - Step 1
  - Step 2
  - …
  - Step N

- Parallel computing takes these steps and does as many at the same time as it can.
  - Step 1 & Step 2 & Step 9
  - Step 3 & Step 5 & Step 7
  - Step 4 & Step 6 & Step 8

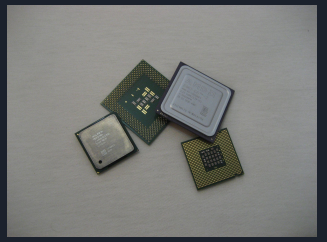
Key Terminology in Parallel Computing

Core

- An individual CPU core on a given computer (most CPUs have multiple cores).

Node

- An individual computer. Often a Node will have many Cores.

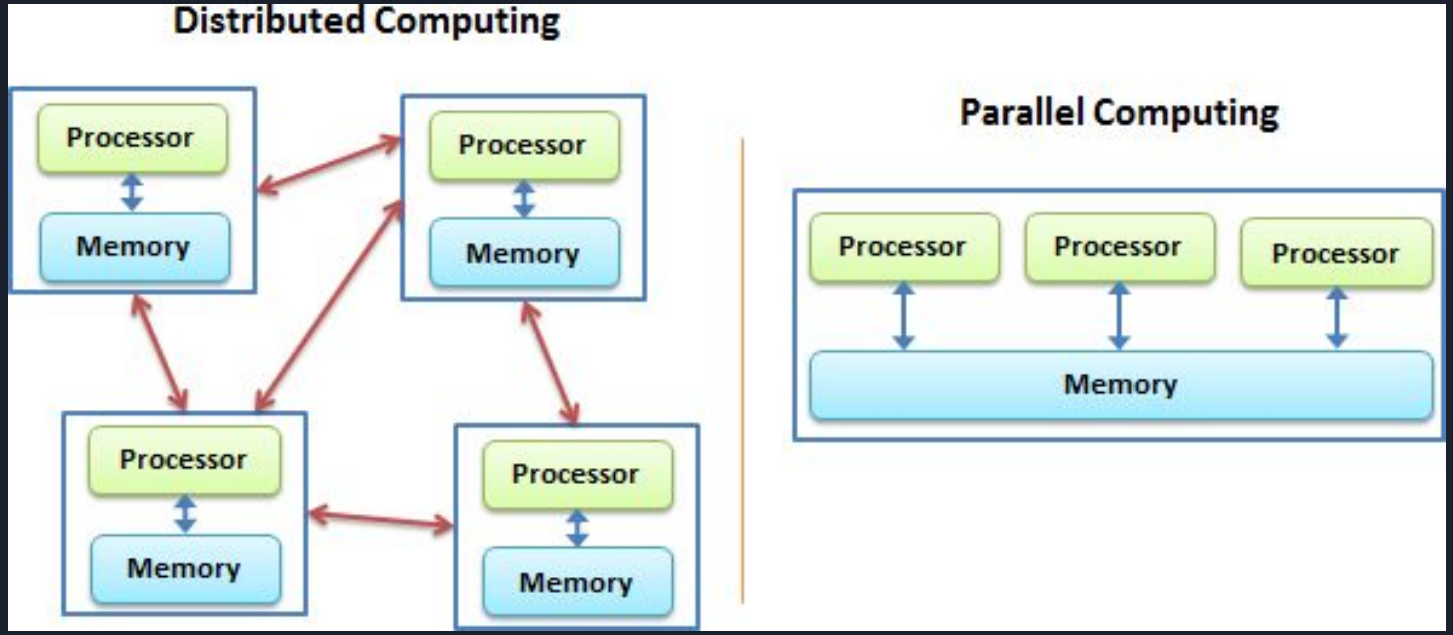


Basic Modalities of Parallel Computing

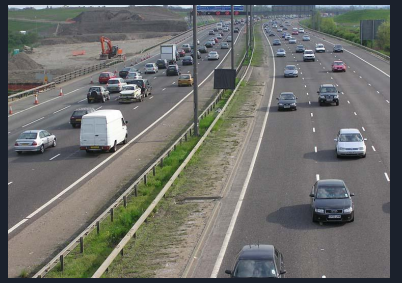
- **Multicore Parallel Computing**
  - Parallel Computing using multiple CPUs and/or multiple CPU Cores.
  - Occurs on a single computer.

- **Distributed Parallel Computing**
  - Parallel Computing using multiple individual computers (nodes) running in tandem with one another.
  - Can leverage all of the individual computers' memory as well.

Basic Modalities of Parallel Computing



Key Terminology in Parallel Computing



Thread

- A very lightweight stream of computations which accesses and writes to global memory.
- All threads access the same pool of memory.
  - In MATLAB, these are faster and less memory intensive but lacks complete support.

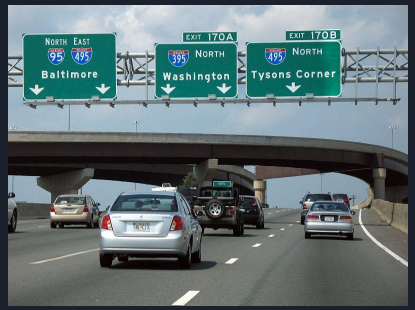
Process

- A relatively isolated channel in which computations can run which has its own memory space. No processes can access any other process's memory space.
- Message Passing is how data is handed between processes.
  - In MATLAB, this is the recommended approach.

Key Terminology in Parallel Computing

SPMD

- "Single Program Multiple Data"
  - A set of code designed to be run multiple times, each on different data.



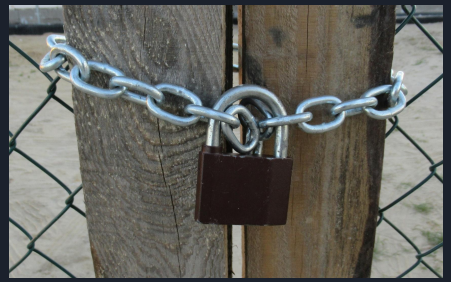
Address/Memory Space

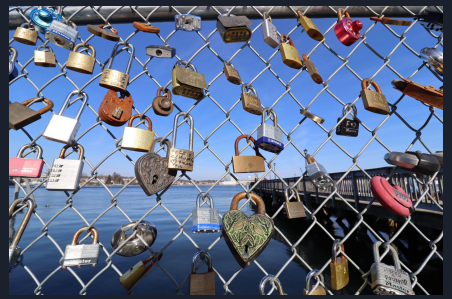
- A set of locations with addresses in the RAM or disk memory of a computer.

Key Terminology in Parallel Computing

Lock

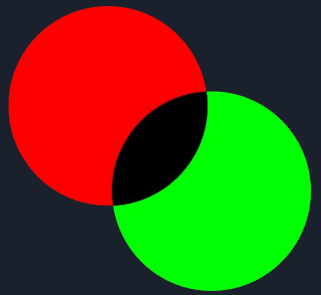
- A "barrier" in the code which requires one or more threads/processes to wait until others have completed.





Mutex

- "Mutually Exclusive" - this is a type of lock which only allows a single process or thread to pass at a time.



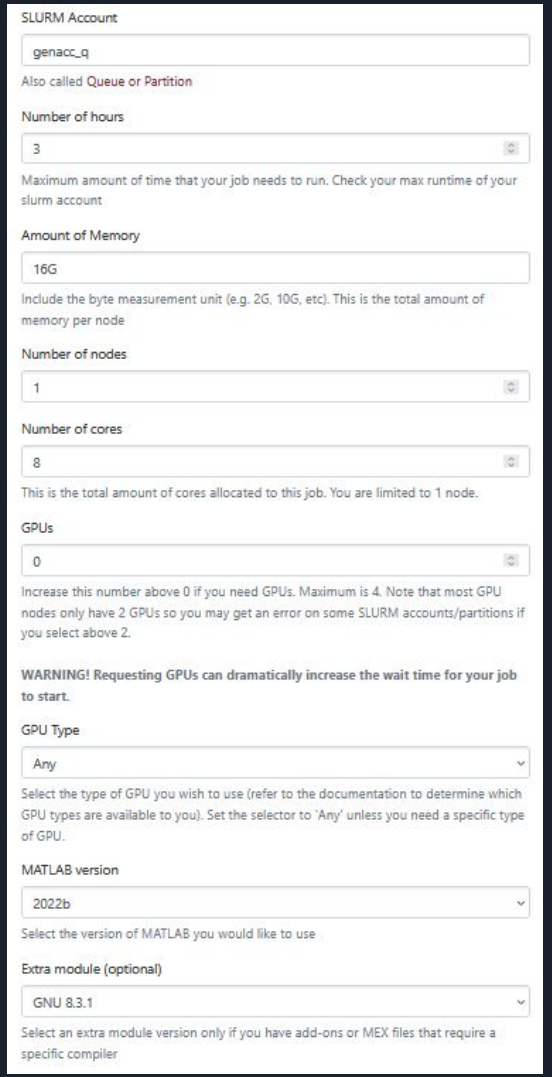
Parallel Computing Paradigms

- **Shared-Memory Parallel Computing**
  - All threads being run share one memory space and can read/write from it at any time.
- **Distributed-Memory Parallel Computing**
  - Each process has its own memory space and must pass messages containing data between processes which require data from another's memory space.
- **PGAS (Partitioned Global Address Space) Parallel Computing**
  - A distributed-memory system that acts like shared memory.
  - A memory space which is physically spread over many nodes but where each item has a unique global address that is known to all processes/threads.

Parallel Computing Setup in MATLAB

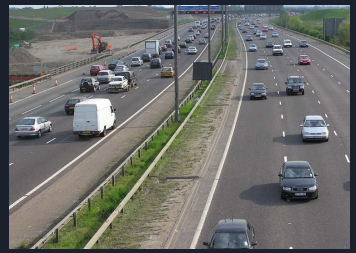
Setting Up an Open OnDemand MATLAB Job

- Select a SLURM Account you have access to: genacc_q (or any other queue you have access to)
- Select the resources you need for your job (total memory for the job, number of cores, GPUs if necessary/available).
- Select the version of MATLAB you want to use. We have several. Note that newer versions will take longer to initialize. This is just due to the JVM requiring more time to initialize expanded features in new versions.



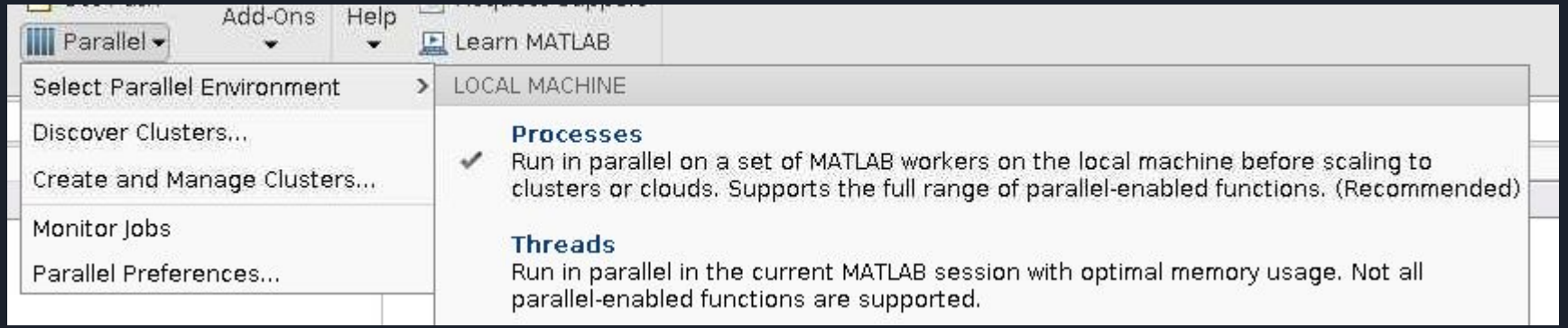
MATLAB and Parallel Computing

- MATLAB can do both Shared-Memory and Distributed-Memory
- Which you end up using depends on the Parallel option you choose in MATLAB's Parallel Environment list.
  - Processes = Distributed-Memory
  - Threads = Shared-Memory



MATLAB and Parallel Computing

- MATLAB can do true distributed computing over multiple nodes…
  - But this requires a separate product on top of the MATLAB Parallel Computing Toolbox.
- Select “Processes” unless you have a reason to do otherwise.



Setting up a Worker Pool in MATLAB

```
% Create a pool of 4 workers using
% the 'local' profile
P = parpool('local', 4);

...


% Remember to shut down the
% parallel pool before closing!
delete(P);
```

- The first **necessary** step for MATLAB parallel computing
- The "parpool" is a group of threads or processes.
- MATLAB calls threads or processes "workers"
- Work is sent from the main process to worker processes.
- The pool can be used for many different parallel computing functionalities.
- **The number of workers must be no greater than the number of cores on the computer you are using!**

Automatic Parallelization using Worker Pools

Functions that Support Automatic Parallelization

```
% Setup your worker pool
P = parpool('local', 4);

…

<function_name>(P, <args>);

<function>(..., 'Parallel', true)

…

% Remember to shut down workers
delete(P);
```

- Several functions in MATLAB come with automatic parallel computing support.
- Some will automatically detect if a parallel pool of workers is active.
- Others will require that you pass the pool handle **P** (in the pseudocode to the left) or they will have a 'UseParallel' or 'Parallel' true/false argument.
- A list of functions that support parallel computing can be found on MATLAB's Function Documentation Site.

Example

```
% Start the parallel pool
P = parpool('local', 4);

% Set up the Bases to select from
bases = ['A', 'C', 'T', 'G'];
% Create a Cell Array of 4 sequences.
S = cell(4,1);
% Generate Sequences
for i = 1:4
    currseq = "";
    for j = 1:10
        currseq = strcat(currseq,bases(randi(numel(bases))));
    end
    S{i} = convertStringsToChars(currseq);
end

% Now, in parallel, compute the sequence
% distances between the 4 sequences
D = seqpdist(S, 'UseParallel', true);

% Plot a heat map of the distances
heatmap(squareform(D));

% Close down the parallel Pool
delete(P);
```

- In this code, we generate 4 random DNA sequences of 10 base pairs.
- Then we generate the pairwise distances between them in parallel using **seqpdist** from the Bioinformatics Toolbox.
- Note the **'UseParallel', true** option.

Parallel Loops in MATLAB

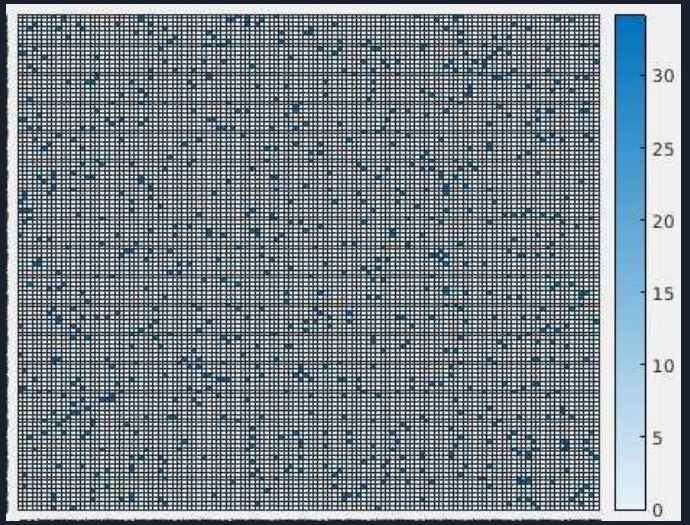
Running Loops in Parallel with `parfor`

```
% Setup your worker pool
P = parpool('local', 4);

...

parfor i = 1:10

    <code_to_run_in_parallel>;

end

...

% Remember to shut down workers
delete(P);
```

- In many cases, you'll want to complete a series of calculations on the same, often large, set of data in parallel.
- For this, the **parfor** loop is a good option.
- Parfor loops work almost exactly like for loops do. They will automatically break up the work into chunks and distribute them to the workers.
- **Be careful of loop dependencies!**
- More information on parfor loops can be found in the documentation.

Example

```
% Start the Parallel Pool
P = parpool('local', 4);
% Set up the Bases to Select from
bases = ['A', 'C', 'T', 'G'];
% Create a Cell Array to Store Sequences
S = cell(120, 1);
% Generate Sequences
parfor i = 1:120
    currseq = "";
    for j = 1:10
        currseq = strcat(currseq, bases(randi(numel(bases))));
    end
    S{i} = convertStringsToChars(currseq);
end
% Now, in Parallel, compute the sequence distances
% between the 120 sequences
D = seqpdist(S, 'UseParallel', true);
% Plot a heatmap of the distances
heatmap(squareform(D));
% Close the Parallel Pool
delete(P);
```



- In this example, we are generating a larger set of small DNA sequences that can be compared using the distance calculation from the first example.

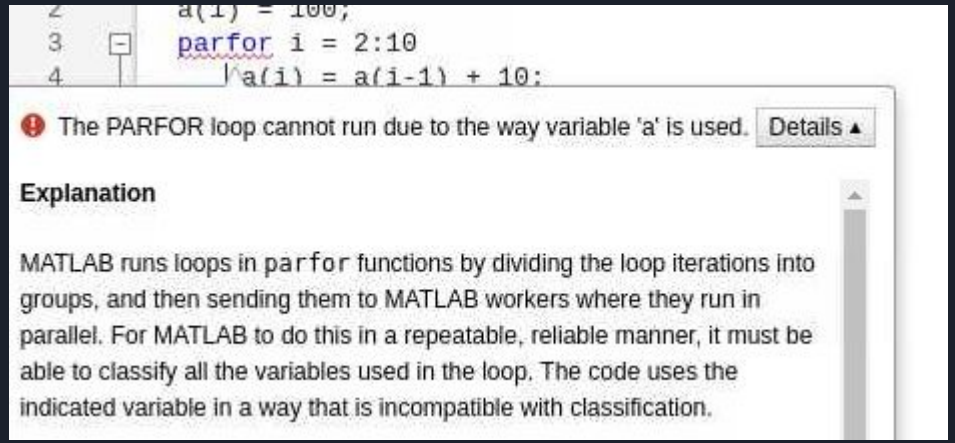
Loop Dependencies

```
for
    dxdt(i) = dxdt(i-1) + (dxdt(i-1)*dt)
end
```

- Occurs when a given iteration of a loop depends on the results of one or more other iterations of the same loop.
- Can result in undefined behavior since you will likely not know whether or not the dependent iterations have completed their execution and properly updated the data by the time the current iteration requires said data.
- It is important to have each iteration of your parallel loop be independent of all others.
- If using nested loops, parallelize only the outermost loop.

A Solution for Common Caveats and Pitfalls in Parallel Computing: The MATLAB Checker

- MATLAB is usually pretty good at catching most of these situations before you run them, though the explanations may be vague. You'll likely get an error about "Unable to classify variable" or something along those lines.



To Recap

- Foundational Parallel Computing Concepts
- Parallel MATLAB Basics
  - Worker Pools
- Parallel Functions in MATLAB
  - Automatic Parallelization Functions
- Do It Yourself Parallelization in MATLAB
  - Parallel Loops (parfor)

Parallel Function Evaluation

Running Functions in Parallel with `parfeval`

```
% Setup your worker pool
P = parpool('local', 4);

…

out = parfeval(P, @myfun, <n>, <args>);

…

% Remember to shut down workers
delete(P);
```

- It's also possible to run other functions in parallel over a set of input arguments.
- The **parfeval** function allows you to do this.

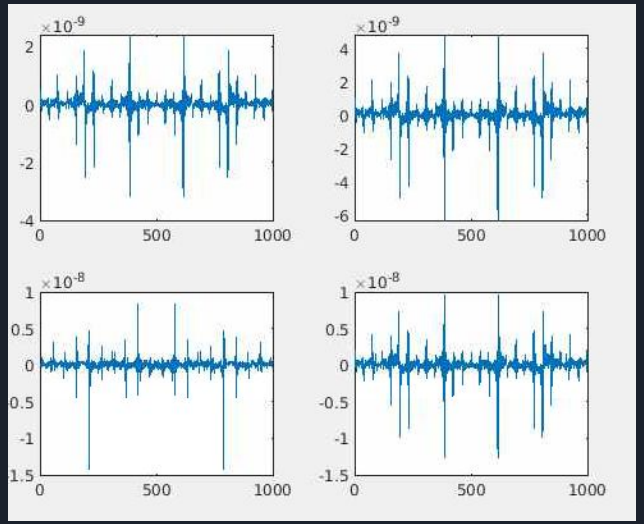
Running Functions in Parallel with `parfeval`

```
% Setup your worker pool
P = parpool('local', 4);

…

out = parfeval(P, @myfun, <n>, <args>);

…

% Remember to shut down workers
delete(P);
```

```
P       - Parallel Pool
@myfun  - Function to run
<n>     - # of Outputs
<args>  - Input arguments

out     - Output
```

Example

```
% Start the parallel pool
P = parpool('local', 4);

% Set up the time range
t = 1:1000;

% In parallel, run a series of ffts on 4 different signals
for i = 1:4
    signal = (2*(i/2)) + (i/4)*sin(2*pi*(i*10)*t) + sin(2*pi*(i*100)*t);
    f(i) = parfeval(P, @fft, 1, signal);
end

% Get outputs
out = zeros(4,1000);
for i = 1:4
    out(i,:) = fetchOutputs(f(i));
end

% Make subplot
figure
subplot(2,2,1), plot(2:1000, out(1,2:1000));
subplot(2,2,2), plot(2:1000, out(2,2:1000));
subplot(2,2,3), plot(2:1000, out(3,2:1000));
subplot(2,2,4), plot(2:1000, out(4,2:1000));

% Close down the parallel Pool
delete(P);
```



- A few good examples can be found on the parfeval documentation page.
- Here, we are taking a simple noisy signal and computing the frequency spectrum of it in parallel.

SPMD Blocks in MATLAB

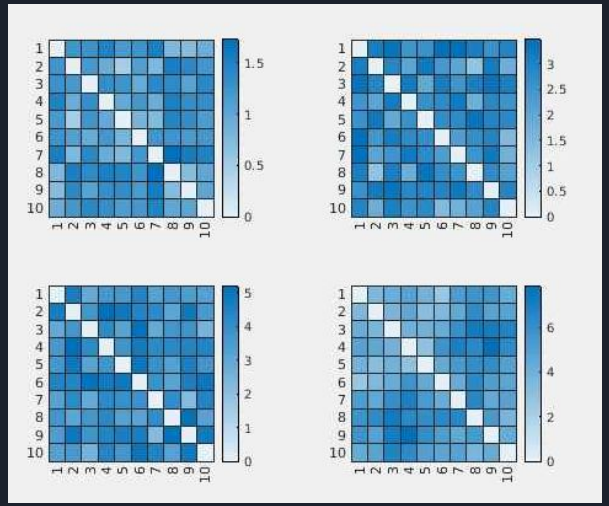
Running Blocks of Code in Parallel with `spmd`

```
% Setup your worker pool
P = parpool(4, 'local');

…
spmd
   <code_to_run_in_parallel>;
end
…
% Remember to shut down workers
delete(P);
```

- SPMD blocks allow you to run the same set of code on multiple different data sets.
- The individual runs are done in parallel.
- This way, you can run the same experiment on different data sets in parallel without having to wait for one to finish before the next one starts.
- Single Program, Multiple Data
- Additional Examples can be found on the spmd documentation page.

Example

```
% Start the parallel pool
P = parpool('local', 4);

% In an SPMD block, generate a Data set and compute it's distance matrix
spmd
    Data = rand(10,10)*spmdIndex;
    Distances = squareform(pdist(Data));
end

% Plot a heat map of the distances
figure
subplot(2,2,1), heatmap(Distances{1});
subplot(2,2,2), heatmap(Distances{2});
subplot(2,2,3), heatmap(Distances{3});
subplot(2,2,4), heatmap(Distances{4});

% Close down the parallel Pool
delete(P);
```



- Here, we use an SPMD block to generate random matrices on-the-fly modulated by the current process index (1, 2, 3 or 4) and compute their pair-wise distances.

Questions?

Links and References

- MATLAB Parallel Computing Toolbox Tutorials
  - https://www.mathworks.com/help/parallel-computing/getting-started-with-parallel-computing-toolbox.html
- MATLAB Parallel Computing Toolbox Documentation
  - https://www.mathworks.com/help/parallel-computing/
- Parallel Computing Workshop Video from MATLAB/MathWorks
  - https://www.mathworks.com/support/search.html/videos/parallel-computing-hands-on-workshop-1594017972362.html?fq%5B%5D=asset_type_name:video&fq%5B%5D=category:matlab-parallel-server/index&page=1
- Parallel Programming Free Online Textbook (C++ and Fortran)
  - https://theartofhpc.com/pcse.html
- The Art of HPC Free Online Textbook Series
  - https://theartofhpc.com/

Thank You!

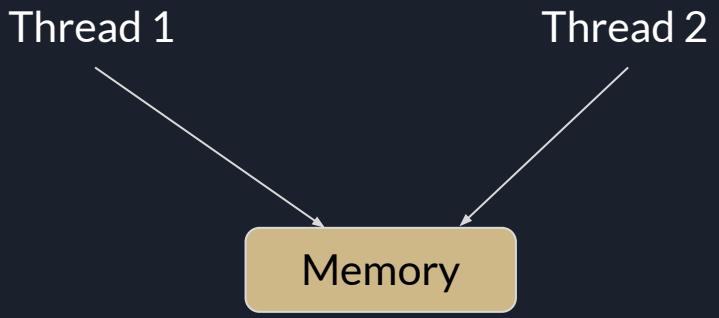
Contact RCC:
support@rcc.fsu.edu

Caveats and Pitfalls in Parallel Computing

- Purpose
  - *Even though MATLAB has some guard rails to prevent issues from occurring, MATLAB will not fix the code for you. It's important to understand what potential issues can occur in parallel computing.*
  - *These caveats are essentially universal to Parallel Computing, they don't just apply to MATLAB.*

- Loop Dependencies
- Race Conditions
- Deadlock

Race Conditions



- Occurs when two more more processes or threads try to update the same location in memory at the same time.
- This can lead to either deadlock or undefined behavior as it becomes impossible to predict which thread will get to update the memory item first.
- It is important to make sure only one process is updating or modifying an item in memory at any given time.

Deadlock

- Deadlock can arise from race conditions and/or loop dependencies.
- Occurs when two or more processes are stuck waiting on each other to finish execution.
- Results in code that appears to "lock" up and stop running.
    - Hence the name "Deadlock"