MATLAB Parallel Computing Toolbox

CSCI 4850/5850 High-Performance Computing

Spring 2019

Tae-Hyuk (Ted) Ahn
Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University
Learning Objectives

- You will learn how to use MATLAB, how to convert the loop in parallel, and various parallel options in MATLAB.

- Remaining topics:
  - Amazon Cloud, Apache Spark
  - GPU (OpenACC, CUDA)
MATLAB is a programming platform designed specifically for engineers and scientists. The heart of MATLAB is the MATLAB language, a matrix-based language allowing the most natural expression of computational mathematics.

What can you do with MATLAB?

Using MATLAB, you can:

- Analyze data
- Develop algorithms
- Create models and applications

The language, apps, and built-in math functions enable you to quickly explore multiple approaches to arrive at a solution. MATLAB lets you take your ideas from research to production by deploying to enterprise applications and embedded devices, as well as integrating with Simulink® and Model-Based Design.

Who uses MATLAB?

Millions of engineers and scientists in industry and academia use MATLAB. You can use MATLAB for a range of applications, including deep learning and machine learning, signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology.
How to Use Parallel Processing?


Key Features

- Parallel for-loops (parfor) for running task-parallel algorithms on multiple processors
- Support for CUDA-enabled NVIDIA GPUs
- Full use of multicore processors on the desktop via workers that run locally
- Computer cluster and grid support (with MATLAB Distributed Computing Server)
- Interactive and batch execution of parallel applications
- Distributed arrays and single program multiple data (spmd) construct for large dataset handling and data-parallel algorithms

Parallel computing with MATLAB. You can use Parallel Computing Toolbox to run applications on a multicore desktop with local workers available in the toolbox, take advantage of GPUs, and scale up to a cluster (with MATLAB Distributed Computing Server).
Goal?

Parallel Computing Toolbox lets you solve computationally and data-intensive problems using MATLAB on multi-core and multiprocessor computers.

The way it does:

Parallel processing constructs such as parallel for-loops and code blocks, distributed arrays, parallel numerical algorithms, and message-passing functions let you implement task- and data-parallel algorithms in MATLAB at a high level without programming for specific hardware and network architectures.
PCT Key Features

- Parallel for-loops (parfor) for running task-parallel algorithms on multiple processors
- Support for CUDA-enabled NVIDIA GPUs
- Full use of multicore processors on the desktop via workers that run locally
- Computer cluster and grid support (with MATLAB Distributed Computing Server)
- Interactive and batch execution of parallel applications
- Distributed arrays and single program multiple data (spmd) construct for large dataset handling and data-parallel algorithms
Why consider parallel computing

- Save time by distributing tasks and executing these simultaneously
- Solve big data problems by distributing data
- Take advantage of your desktop computer resources and scale up to clusters and cloud computing
Parallel Computing Toolbox

- Accelerate your code using interactive parallel computing tools, such as `parfor` and `parfeval`.
- Scale up your computation using interactive Big Data processing tools, such as `distributed`, `tall`, `datastore`, and `mapreduce`.
- Use `gpuArray` to speed up your calculation on the GPU of your computer.
- Use `batch` to offload your calculation to computer clusters or cloud computing facilities.
Parallel Computing concepts

- CPU: Central Processing Unit, comprising multiple cores or processors
- GPU: Graphics Processing Unit, now widely used for general purpose (GP) GPU computing
- Node: standalone computer, containing one or more CPUs / GPUs. Nodes are networked to form a cluster or supercomputer
- Thread: smallest set of instructions that can be managed independently by a scheduler. On a GPU, multiprocessor or multicore system, multiple threads can be executed simultaneously (multi-threading)
- Batch: off-load execution of a functional script to run in the background
- Scalability: increase in parallel speedup with the addition of more resources
What tools Parallel Computing Toolbox offer

- MATLAB workers: MATLAB computational engines for parallel computing, associated with the cores in a multicore machine
- Parallel pool: a parallel pool of MATLAB workers can be created using `parpool`
- Speed up: Accelerate your code by running on multiple MATLAB workers, using `parfor` and `parfeval`
- Scale up: Partition your big data across multiple MATLAB workers, using `distributed arrays` and `mapreduce`
- Asynchronous processing: Use `parfeval` to execute a computing task without waiting for it to complete
## Choose a Parallel Computing Solution

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solutions</th>
<th>Required Products</th>
<th>More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you want to process your data faster?</td>
<td>Profile your code.</td>
<td>MATLAB</td>
<td>Profile to Improve Performance (MATLAB)</td>
</tr>
<tr>
<td></td>
<td>Vectorize your code.</td>
<td>MATLAB</td>
<td>Vectorization (MATLAB)</td>
</tr>
<tr>
<td></td>
<td>Use built-in parallel computing support in MathWorks products.</td>
<td>MATLAB Parallel Computing Toolbox</td>
<td>Built-in Parallel Computing Support</td>
</tr>
<tr>
<td></td>
<td>If you have a GPU, try gpuArray.</td>
<td>MATLAB Parallel Computing Toolbox</td>
<td>Identify and Select a GPU Device</td>
</tr>
<tr>
<td></td>
<td>Use parfor.</td>
<td>MATLAB Parallel Computing Toolbox</td>
<td>Interactively Run a Loop in Parallel Using parfor</td>
</tr>
<tr>
<td>Are you looking for other ways to speed up your processing?</td>
<td>Try parfeval.</td>
<td>MATLAB Parallel Computing Toolbox</td>
<td>Evaluate Functions in the Background Using parfeval</td>
</tr>
<tr>
<td></td>
<td>Try spmd.</td>
<td>MATLAB Parallel Computing Toolbox</td>
<td>Run Single Programs on Multiple Data Sets</td>
</tr>
<tr>
<td>Do you want to scale up your big data calculation?</td>
<td>To work with out-of-memory data with any number of rows, use tall arrays.</td>
<td>MATLAB</td>
<td>Big Data Workflow Using Tall Arrays and Datasets</td>
</tr>
<tr>
<td></td>
<td>This workflow is well suited to data analytics and machine learning.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use tall arrays in parallel on your local machine.</td>
<td>MATLAB Parallel Computing Toolbox</td>
<td>Use Tall Arrays on a Parallel Pool</td>
</tr>
<tr>
<td></td>
<td>Use tall arrays in parallel on your cluster.</td>
<td>MATLAB Parallel Computing Toolbox</td>
<td>Use Tall Arrays on a Spark Enabled Hadoop Cluster</td>
</tr>
<tr>
<td></td>
<td>If your data is large in multiple dimensions, use distributed instead.</td>
<td>MATLAB Parallel Computing Toolbox MATLAB Distributed Computing Server</td>
<td>Distributing Arrays</td>
</tr>
<tr>
<td></td>
<td>This workflow is well suited to linear algebra problems.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Do you want to offload to a cluster?</td>
<td>Use batch to run your code on clusters and clouds.</td>
<td>MATLAB Distributed Computing Server</td>
<td>Run Batch Parallel Jobs</td>
</tr>
</tbody>
</table>
In this example, you start with a slow for-loop, and you speed up the calculation using a parfor-loop instead. `parfor` splits the execution of for-loop iterations over the workers in a parallel pool.
Interactively Run a Loop in Parallel Using parfor

1. In the MATLAB® Editor, enter the following for-loop. Add tic and toc to measure the time elapsed.

```matlab
tic
n = 200;
A = 500;
a = zeros(n);
for i = 1:n
    a(i) = max(abs(eig(rand(A))));
end
toc
```

2. Run the script, and note the elapsed time.

   Elapsed time is 31.935373 seconds.
Interactively Run a Loop in Parallel Using \texttt{parfor}

3. In the script, replace the for-loop with a \texttt{parfor}-loop.

```matlab
 tic
 n = 200;
 A = 500;
 a = zeros(n);
 parfor i = 1:n
   a(i) = max(abs(eig(rand(A))));
 end
 toc
```

4. Run the new script, and run it \texttt{again}. Note that the \texttt{first run is slower than the second run}, because the parallel pool takes some time to start and make the code available to the workers. Note the elapsed time for the second run.
Ensure That parfor-Loop Iterations are Independent

- If you get an error when you convert for-loops to parfor-loops, ensure that your parfor-loop iterations are independent.
- parfor-loop iterations have no guaranteed order, while the iteration order in for-loops is sequential.
- Also parfor-loop iterations are performed on different MATLAB® workers in the parallel pool, so that there is no sharing of information between iterations.
- Therefore one parfor-loop iteration must not depend on the result of a previous iteration.
- The only exception to this rule is to accumulate values in a loop using Reduction Variables.
Ensure That parfor-Loop Iterations are Independent

clear A
d = 0; i = 0;
for i = 1:4
    d = i+2;
    A(i) = d;
end
A
d
i

A =
   2  4  6  8
d =
   8
i =
   4

clear A
d = 0; i = 0;
parfor i = 1:4
    d = i+2;
    A(i) = d;
end
A
d
i

A =
   2  4  6  8
d =
   0
i =
   0
Nested `parfor`-Loops and `for`-Loops

- You cannot use a `parfor`-loop inside another `parfor`-loop. As an example, the following nesting of `parfor`-loops is **not allowed**:

  ```matlab
  parfor i = 1:10
    parfor j = 1:5
      ...
    end
  end
  ```

- Parallel processing incurs overhead. Generally, you should run the outer loop in parallel, because overhead only occurs once. If you run the inner loop in parallel, then each of the multiple `parfor` executions incurs an overhead.

- Make sure that the number of iterations exceeds the number of workers. Otherwise, you do not use all available workers.

- Try to balance the `parfor`-loop iteration times. `parfor` tries to compensate for some load imbalance.
MATLABPOOL / PARPOOL / GCP

- The `parpool` command replaced the `matlabpool` command in version R2013b.
- Since the change from `matlabpool` to `parpool`, there is an even easier way to create the pool. Unlike `parpool`, it doesn't throw an error if the pool already exists. Just call `gcp` (which stands for "get current pool").

```matlab
gep();
A = zeros(1,10000000);
parfor i = 1:length(A)
    A(i) = i;
end
```
How to check number of cores?

```python
>> feature('numcores')
```
Run Batch Jobs

1. To create the script, type:

   ```matlab
edit mywave```

2. In the MATLAB Editor, create a for-loop:

   ```matlab
   for i = 1:1024
       A(i) = sin(i^2*pi/1024);
   end```

3. Save the file and close the Editor.

4. Use the `batch` command in the MATLAB Command Window to run your script on a separate MATLAB worker:

   ```matlab
   job = batch('mywave')```

5. The `batch` command does not block MATLAB, so you must wait for the job to finish before you can retrieve and view its results:

   ```matlab
   wait(job)```

6. The `load` command transfers variables created on the worker to the client workspace, where you can view the results:

   ```matlab
   load(job,'A')
   plot(A)```

7. When the job is complete, permanently delete its data and remove its reference from the workspace:

   ```matlab
   delete(job)
clear job```
Run Batch Parallel Jobs

1. To create a script, type:
   ```matlab
edit mywave```

2. In the MATLAB Editor, create a `parfor`-loop:
   ```matlab
   parfor i = 1:1024
       A(i) = sin(1^2*pi/1024);
   end
   ```

3. Save the file and close the Editor.

4. Run the script in MATLAB with the `batch` command. Indicate that the script should use a parallel pool for the loop:
   ```matlab
   job = batch('mywave','Pool',3)
   ```
   This command specifies that three workers (in addition to the one running the batch script) are to evaluate the loop iterations. Therefore, this example uses a total of four local workers, including the one worker running the batch script. Altogether, there are five MATLAB sessions involved, as shown in the following diagram.

   ![Diagram of MATLABBatchProcessing]

5. To view the results:
   ```matlab
   wait(job)
   load(job,'A')
   plot(A)
   ```
>> pmode start local  % assuming 4 workers are available
pmode

>> pmode start local % use 4 workers in parallel mode

PCT terminologies:
worker = processor
labindex:
processor number
numlabs:
Number of processors

- Any command issued at the “P>>” prompt is executed on all workers. Enter “labindex” to query for workers’ ID.
- Use if conditional with labindex to issue instructions to specific workers, like this: if labindex==1, numlabs, end
• **Replicate array**
  \[ P>> A = \text{magic}(3); \quad \% \text{A is replicated on every worker} \]

• **Variant array**
  \[ P>> A = \text{magic}(3) + \text{labindex} - 1; \quad \% \text{labindex=1,2,3,4} \]

<table>
<thead>
<tr>
<th>LAB 1</th>
<th>LAB 2</th>
<th>LAB 3</th>
<th>LAB 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

• **Private array**
  \[ P>> \text{if labindex==2, A = magic}(3) + \text{labindex} - 1; \text{end} \]

<table>
<thead>
<tr>
<th>LAB 1</th>
<th>LAB 2</th>
<th>LAB 3</th>
<th>LAB 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>undefined</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>undefined</td>
<td>5</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>
Additional Resources

- Parallel Computing Toolbox
  