Parallel Computing with MATLAB

Jiro Doke, Ph.D.
Senior Application Engineer
The MathWorks, Inc.
Agenda

- Speed up algorithms without code changes
- Develop parallel code interactively
  - Task Parallel
  - Data Parallel
- Schedule your programs to run offline
- Operational Requirements
Some Questions to Consider

- Do you want to speed up your algorithms?
- Do you have datasets too big to fit on your computer?

If so…

- Do you have a multicore or multiprocessor desktop machine?
- Do you have access to a computer cluster?
Utilizing Additional Processing Power

- Built-in multithreading
  - Core MATLAB
  - Introduced in R2007a
  - Utility for specific matrix operations
  - Automatically enabled since R2008a

- Parallel computing tools
  - Parallel Computing Toolbox
  - MATLAB Distributed Computing Server
  - Broad utility controlled by the MATLAB user
Implicit Multithreaded Computations

- No change required for user code
- Enables multithreading for key mathematical routines
  - Linear algebra operations
  - Element-wise operations
Implicit Multithreaded Computation

- Multithreaded Libraries
  - Multithreaded Basic Linear Algebra Subroutines (BLAS)
    - BLAS are vendor specific
    - Optimized for specific processor

- Element-wise operations
  - Just-in-time acceleration (JIT) generates on-the-fly multithreaded code
## Solving Big Technical Problems

### Challenges

| Long running | Computationally intensive |

### You could…

- Wait

### Solutions

- Run similar *tasks* on independent processors in *parallel*

- Load *data* onto multiple machines that work together in *parallel*
Going Beyond Serial MATLAB Applications
Programming Parallel Applications

Level of control

Minimal
Some
Extensive

Level of effort

None
Straightforward
Extensive
Programming Parallel Applications

Level of effort

- None
- Straightforward
- Extensive

Support built into Toolboxes and Blocksets
Example: Optimizing Tower Placement

- Determine location of cell towers
- Maximize coverage
- Minimize overlap
Summary of Example

- Enabled built-in support for Parallel Computing Toolbox in Optimization Toolbox
- Used a pool of MATLAB workers
- Optimized in parallel using `fmincon`
Parallel Computing Support in Optimization Toolbox

- **Functions:**
  - `fmincon`:
    - Finds a constrained minimum of a function of several variables
  - `fminimax`:
    - Finds a minimax solution of a function of several variables
  - `fgoalattain`:
    - Solves the multiobjective goal attainment optimization problem

- Functions can take finite differences in parallel in order to speed the estimation of gradients
Tools Providing Parallel Computing Support

- Optimization Toolbox
- GADS Toolbox
- Statistics Toolbox
- SystemTest
- Simulink Design Optimization
- Bioinformatics Toolbox
- Model-Based Calibration Toolbox
- …

*Directly leverage functions in Parallel Computing Toolbox*
Programming Parallel Applications

Level of effort

None

Support built into Toolboxes and Blocksets

Straightforward

Simple programming constructs:

parfor

Extensive
Task Parallel Applications

- **MATLAB**
- **SIMULINK**
- **TOOLBOXES**
- **BLOCKSETS**

Worker
Worker
Worker
Worker

Task 1  Task 2  Task 3  Task 4

Time
Example: Parameter Sweep of ODEs

- Solve a 2\textsuperscript{nd} order ODE

\[
m \ddot{x} + b \dot{x} + k x = 0
\]

- Simulate with different values for \( b \) and \( k \)

- Record peak value for each run

- Plot results
Summary of Example

- Mixed task-parallel and serial code in the same function
- Ran loops on a pool of MATLAB resources
- Used M-Lint analysis to help in converting existing for-loop into parfor-loop
The Mechanics of `parfor` Loops

```matlab
a = zeros(10, 1);
parfor i = 1:10
    a(i) = i;
end
```

Pool of MATLAB Workers
Converting for to parfor

- Requirements for parfor loops
  - Task independent
  - Order independent

- Constraints on the loop body
  - Cannot “introduce” variables (e.g. eval, load, global, etc.)
  - Cannot contain break or return statements
  - Cannot contain another parfor loop
Advice for Converting for to parfor

- Use M-Lint to diagnose parfor issues

- If your for loop cannot be converted to a parfor, consider wrapping a subset of the body to a function

- Read the section in the documentation on classification of variables
Parallel Computing Tools Address…

Long computations

- Multiple independent iterations
  
  \[
  \text{parfor } i = 1 : n \\
  \text{\% do something with } i \\
  \text{end}
  \]

- Series of tasks

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
<th>Task 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>26</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>27</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>28</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>29</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>30</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>31</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>32</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>33</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>34</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>35</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>36</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>37</td>
<td>52</td>
<td></td>
</tr>
</tbody>
</table>

Large data problems
Large Data Sets (Data Parallel)
Limited Process Memory

- **32-bit platforms**
  - Windows 2000 and XP (by default): 2 GB
  - Linux/UNIX/MAC system configurable: 3-4 GB
  - Windows XP with /3gb boot.ini switch: 3 GB

- **64-bit platforms**
  - Linux: 8 TB
  - Windows XP Professional x64: 8TB
Programming Parallel Applications

Level of effort

None

Support built into Toolboxes and Blocksets

Straightforward

Simple programming constructs:

parfor
distributed

Extensive
Client-side Distributed Arrays

Remotely Manipulate Array from Desktop

Distributed Array Lives on the Cluster
Distributed Arrays and Parallel Algorithms

- Distributed arrays
  - Store segments of data across participating workers
  - Create from any built-in class in MATLAB
    - Examples: doubles, sparse, logicals, cell arrays, and arrays of structures

- Parallel algorithms for distributed arrays
  - Matrix manipulation operations
    - Examples: indexing, data type conversion, and transpose
  - Parallel linear algebra functions, such as `svd` and `lu`
  - Data distribution
    - Automatic, specify your own, or change at any time
## Enhanced MATLAB Functions That Operate on Distributed Arrays

<table>
<thead>
<tr>
<th>Type of Function</th>
<th>Function Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data functions</td>
<td><code>cumprod, cumsum, fft, max, min, prod, sum</code></td>
</tr>
<tr>
<td>Data type functions</td>
<td><code>cast, cell2mat, cell2struct, celldisp, cellfun, char, double, fieldnames, int16, int32, int64, int8, logical, num2cell, rmfield, single, struct2cell, swapbytes, typecast, uint16, uint32, uint64, uint8</code></td>
</tr>
<tr>
<td>Elementary and trigonometric functions</td>
<td><code>abs, acos, acosd, acosh, acot, acotd, acoth, acsc, acscd, acsch, angle, asec, asecd, asech, asin, asind, asinh, atan, atan2, atand, atanh, ceil, complex, conj, cos, cosd, cosh, cot, cotd, coth, csc, csd, csch, exp, expm1, fix, floor, hypot, imag, isreal, log, log10, log1p, log2, mod, nextpow2, nthroot, pow2, real, reallog, realpow, realsqrt, rem, round, sec, secd, sech, sign, sin, sind, sinh, sqrt, tan, tand, tanh</code></td>
</tr>
<tr>
<td>Elementary matrices</td>
<td><code>cat, diag, eps, find, isempty, isequal, isequalwithnan, isfinite, isinf, isnan, length, ndims, size, tril, triu</code></td>
</tr>
<tr>
<td>Matrix functions</td>
<td><code> chol, eig, lu, norm, normest, svd</code></td>
</tr>
<tr>
<td>Array operations</td>
<td><code>all, and, any, bitand, bitor, bitxor, ctranspose, end, eg, ge, gt, horzcat, ldivide, le, lt, minus, mldivide, mrdivide, mtimes, ne, not, or, plus, power, rdivide, subsasgn, subsindex, subsref, times, transpose, uminus, uplus, vertcat, xor</code></td>
</tr>
<tr>
<td>Sparse matrix functions</td>
<td><code>full, issparse, nnz, nonzeros, nzmax, sparse, spfun, spzeros</code></td>
</tr>
<tr>
<td>Special functions</td>
<td><code>dot</code></td>
</tr>
</tbody>
</table>
Client-side Distributed Arrays and SPMD

- **Client-side distributed arrays**
  - Class `distributed`
  - Can be created and manipulated directly from the client.
  - Simpler access to memory on labs
  - Client-side visualization capabilities

- `spmd`
  - Block of code executed on workers
  - Worker specific commands
  - Explicit communication between workers
  - Mixture of parallel and serial code
**spmd blocks (Data Parallel)**

```
spmd
  % single program across workers
end
```

- Mix data-parallel and serial code in the same function
- Run on a pool of MATLAB resources
- **Single Program** runs simultaneously across workers
  - Distributed arrays, message-passing
- **Multiple Data** spread across multiple workers
  - Data stays on workers
A mental model for SPMD … END

Pool of MATLAB Workers
MPI-Based Functions in Parallel Computing Toolbox™

Use when a high degree of control over parallel algorithm is required

- High-level abstractions of MPI functions
  - `labSendReceive`, `labBroadcast`, and others
  - Send, receive, and broadcast any data type in MATLAB

- Automatic bookkeeping
  - Setup: communication, ranks, etc.
  - Error detection: deadlocks and miscommunications

- Pluggable
  - Use any MPI implementation that is *binary*-compatible with MPICH2
Scheduling Applications
Interactive to Scheduling

- Interactive
  - Great for prototyping
  - Immediate access to MATLAB workers

- Scheduling
  - Offloads work to other MATLAB workers (local or on a cluster)
  - Access to more computing resources for improved performance
  - Frees up local MATLAB session
Scheduling Work

MATLAB

SIMULINK

TOOLBOXES

BLOCKSETS

Worker

Worker

Worker

Scheduler

Work

Result
Example: Schedule Processing

- Offload parameter sweep to local workers
- Get peak value results when processing is complete
- Plot results in local MATLAB
Summary of Example

- Used `batch` for off-loading work
- Used `matlabpool` option to off-load and run in parallel
- Used `load` to retrieve worker’s workspace
Task-Parallel Workflows

- **parfor**
  - Multiple independent iterations
  - Easy to combine serial and parallel code
  - Workflow
    - Interactive using `matlabpool`
    - Scheduled using `batch`

- **jobs/tasks**
  - Series of independent tasks; not necessarily iterations
  - Workflow ➔ Always scheduled
Programming Parallel Applications

Level of effort

None  
Support built into Toolboxes and Blocksets

Straightforward

Simple programming constructs:
parfor

Extensive

Full control of parallelization:
job and task
Scheduling Jobs and Tasks

MATLAB

SIMULINK

TOOLBOXES

BLOCKSETS

Scheduler

Worker

Task

Result

Job

Results
Example: Scheduling Independent Simulations

- Offload three independent approaches to solving our previous ODE example

- Retrieve simulated displacement as a function of time for each simulation

- Plot comparison of results in local MATLAB
Summary of Example

- Used `findResource` to find scheduler
- Used `createJob` and `createTask` to set up the problem
- Used `submit` to off-load and run in parallel
- Used `getAllOutputArguments` to retrieve all task outputs
Options for Scheduling Jobs

<table>
<thead>
<tr>
<th>Task Parallel</th>
<th>Data Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&gt;&gt; createMatlabPoolJob</code></td>
<td><code>&gt;&gt; createMatlabPoolJob</code></td>
</tr>
<tr>
<td>or <code>batch</code></td>
<td>or <code>batch</code></td>
</tr>
<tr>
<td><code>&gt;&gt; createJob(...)</code></td>
<td><code>&gt;&gt; createParallelJob</code></td>
</tr>
<tr>
<td><code>&gt;&gt; createTask(...)</code></td>
<td><code>&gt;&gt; createTask(...)</code></td>
</tr>
</tbody>
</table>
Factors to Consider for Scheduling

- There is always an overhead to distribution
  - Combine small repetitive function calls

- Share code and data with workers efficiently
  - Set job properties (FileDependencies, PathDependencies)

- Minimize I/O
  - Enable Workspace option for batch

- Capture command window output
  - Enable CaptureDiary option for batch
Parallel Computing with MATLAB

- Built in parallel functionality within specific toolboxes (also requires Parallel Computing Toolbox)

- High level parallel functions

- Low level parallel functions

- Built on industry standard libraries
Run 8 Local Workers on Desktop

- Rapidly develop parallel applications on local computer
- Take full advantage of desktop power
- Separate computer cluster not required
Scale Up to Clusters, Grids and Clouds
Support for Schedulers

Direct Support

Open API for others
Programming Parallel Applications

**Level of effort**

- **None**: Support built into Toolboxes and Blocksets
- **Straightforward**: Simple programming constructs: `parfor` and `distributed`
- **Extensive**: Full control of parallelization: `job` and `task`, `labSendReceive`