Claude TADONKI and Pierre-Louis CARUANA
Mines ParisTech – CRI (Fontainebleau/France)
University of Paris-Sud (Orsay/France)
Motivations

⭐ MATLAB is widely used for several kinds of application (scientific computing, image processing, ...)
⭐ MATLAB provides a programming language suitable for ordinary scientist (not programmers!)
⭐ MATLAB is commonly used for heavy computations (simulations, image & signal processing)
⭐ Multicore architecture is now a standard, with an increasing number of cores.
⭐ MATLAB offers a built-in solution for parallel computing through additional packages.

Providing a flexible way to consider parallelism in MATLAB is really useful to easily take advantage of this possibility.
MATLAB Solutions for Parallel Computing (Tasks Feature)

We now state some important facts:

- each task within a job is assigned to a unique MATLAB worker, and is executed independently of the other tasks
- the maximum number of workers is specified in the local scheduler profile, and can be modified as desired, up to a limit of twelve
- if a job has more tasks than allowed workers, the scheduler waits for one of the active tasks to complete before starting another MATLAB worker for the next task. In some cases, such an overloading will prevent the entire job from being executed.

% Create one job object
j = createJob();
% Create tasks for job j
createTask(j, @sum, 1, {U(1:4)});
createTask(j, @sum, 1, {U(5:8)});
% Submit job j to the scheduler
submit(j);
% Wait for job completion
wait(j);
% Get the outputs of job j
v = fetchOutputs(j);
% Aggregate the partial sums
s = v{1} + v{2};
% Delete job j
delete(j);

- **jobs/tasks**
  - Series of independent tasks; not necessarily iterations
  - Workflow: Always scheduled

Seamless Parallelism in MATLAB by Claude TADONKI & Pierre-Louis CARUANA
Parallel and Distributed Computing and Networks (PDCN 2014) – Feb 17-19, Innsbruck (AUSTRIA)
MATLAB Solutions for Parallel Computing (Parfor)

- **parfor**
  - Multiple independent iterations
  - Easy to combine serial and parallel code

The **parfor** construct is used, in place of a standard `for` statement, to specify that the corresponding loop should be executed in parallel.

```matlab
p = 1; t = 7;
u = rand(20,1);
parfor i=1:20
  x = 2*i^2 + 1;
p = p*i;
v(i) = u(i) + p;
  if(p <= t)
      v(i) = sqrt(x);
  end
end
```

Mapping a flow of instructions to a **parfor**

Variable kinds within a **parfor** loop

---

Seamless Parallelism in MATLAB by Claude TADONKI & Pierre-Louis CARUANA
Parallel and Distributed Computing and Networks (PDCN 2014) – Feb 17-19, Innsbruck (AUSTRIA)
Our solution (MATLAB engine & Pthread)

We run a C mex-file which creates and launch the threads and manages all data transfers.

Each thread call a MATLAB engine which executes the associated MATLAB instruction.

On WINDOWS, the Automation mode allow to avoid opening a new MATLAB each time we call the engine.

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>engOpen</td>
<td>Start up MATLAB engine</td>
</tr>
<tr>
<td>engClose</td>
<td>Shut down MATLAB engine</td>
</tr>
<tr>
<td>engGetVariable</td>
<td>Get a MATLAB array from the engine</td>
</tr>
<tr>
<td>engPutVariable</td>
<td>Send a MATLAB array to the engine</td>
</tr>
<tr>
<td>engEvalString</td>
<td>Execute a MATLAB command</td>
</tr>
<tr>
<td>engOutputBuffer</td>
<td>Create a buffer to store MATLAB text output</td>
</tr>
<tr>
<td>engOpenSingleUse</td>
<td>Start a MATLAB engine, nonshared use</td>
</tr>
<tr>
<td>engGetVisible</td>
<td>Determine visibility of MATLAB engine session</td>
</tr>
<tr>
<td>engSetVisible</td>
<td>Show or hide MATLAB engine session</td>
</tr>
</tbody>
</table>
Illustrations and Performances

Table 1. Pure overhead of our mechanism

<table>
<thead>
<tr>
<th>run</th>
<th>pthread(s)</th>
<th>task(s)</th>
<th>parfor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.8228</td>
<td>5.9950</td>
<td>9.4820</td>
</tr>
<tr>
<td>2</td>
<td>4.9977</td>
<td>5.9581</td>
<td>9.4874</td>
</tr>
<tr>
<td>3</td>
<td>5.9762</td>
<td>5.9286</td>
<td>9.0390</td>
</tr>
<tr>
<td>4</td>
<td>4.9950</td>
<td>5.9685</td>
<td>8.9879</td>
</tr>
<tr>
<td>5</td>
<td>4.9103</td>
<td>5.9410</td>
<td>9.0397</td>
</tr>
</tbody>
</table>

Table 2. Time costs for data import & export

<table>
<thead>
<tr>
<th>(l_{vector})</th>
<th>pthread(s)</th>
<th>task(s)</th>
<th>parfor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10^6)</td>
<td>0.144</td>
<td>0.687</td>
<td>0.122</td>
</tr>
<tr>
<td>(2 \times 10^6)</td>
<td>0.640</td>
<td>1.407</td>
<td>0.898</td>
</tr>
<tr>
<td>(3 \times 10^6)</td>
<td>0.946</td>
<td>2.114</td>
<td>1.607</td>
</tr>
<tr>
<td>(4 \times 10^6)</td>
<td>1.332</td>
<td>3.777</td>
<td>2.205</td>
</tr>
<tr>
<td>(5 \times 10^6)</td>
<td>1.713</td>
<td>6.604</td>
<td>2.413</td>
</tr>
</tbody>
</table>

- The time overheads provided do not depend on how heavy is the associated task.
- Pthreads based solution has the lowest overhead and is more stable.
- The cost for data import & export suggests that we better use them intensively.
Illustrations and Performances

**Matrix-Product**

<table>
<thead>
<tr>
<th></th>
<th>Pthread</th>
<th>Tasks</th>
<th>Parfor</th>
</tr>
</thead>
<tbody>
<tr>
<td>400x400</td>
<td>0.12</td>
<td>0.16</td>
<td>0.12</td>
</tr>
<tr>
<td>800x800</td>
<td>0.74</td>
<td>0.82</td>
<td>0.69</td>
</tr>
<tr>
<td>1200x1200</td>
<td>1.34</td>
<td>1.22</td>
<td>1.25</td>
</tr>
<tr>
<td>1600x1600</td>
<td>1.64</td>
<td>1.52</td>
<td>1.6</td>
</tr>
<tr>
<td>2000x2000</td>
<td>1.81</td>
<td>1.68</td>
<td>1.76</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Pthread</th>
<th>Tasks</th>
<th>Parfor</th>
</tr>
</thead>
<tbody>
<tr>
<td>400x400</td>
<td>0.11</td>
<td>0.3</td>
<td>0.19</td>
</tr>
<tr>
<td>800x800</td>
<td>0.89</td>
<td>1.27</td>
<td>1.06</td>
</tr>
<tr>
<td>1200x1200</td>
<td>1.67</td>
<td>1.95</td>
<td>1.8</td>
</tr>
<tr>
<td>1600x1600</td>
<td>2.42</td>
<td>2.42</td>
<td>2.56</td>
</tr>
<tr>
<td>2000x2000</td>
<td>2.79</td>
<td>2.56</td>
<td>2.76</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Pthread</th>
<th>Tasks</th>
<th>Parfor</th>
</tr>
</thead>
<tbody>
<tr>
<td>400x400</td>
<td>0.1</td>
<td>0.3</td>
<td>0.35</td>
</tr>
<tr>
<td>800x800</td>
<td>0.99</td>
<td>1.27</td>
<td>1.48</td>
</tr>
<tr>
<td>1200x1200</td>
<td>2.13</td>
<td>1.91</td>
<td>2.18</td>
</tr>
<tr>
<td>1600x1600</td>
<td>3.01</td>
<td>2.42</td>
<td>2.84</td>
</tr>
<tr>
<td>2000x2000</td>
<td>3.01</td>
<td>2.65</td>
<td>2.91</td>
</tr>
</tbody>
</table>

**Sorting**

<table>
<thead>
<tr>
<th></th>
<th>Pthread</th>
<th>Tasks</th>
<th>Parfor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>1.42</td>
<td>1.21</td>
<td>1.13</td>
</tr>
<tr>
<td>2000000</td>
<td>1.63</td>
<td>1.32</td>
<td>1.32</td>
</tr>
<tr>
<td>3000000</td>
<td>1.65</td>
<td>1.33</td>
<td>1.32</td>
</tr>
<tr>
<td>4000000</td>
<td>1.61</td>
<td>1.28</td>
<td>1.28</td>
</tr>
<tr>
<td>5000000</td>
<td>1.82</td>
<td>1.43</td>
<td>1.43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Pthread</th>
<th>Tasks</th>
<th>Parfor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>1.94</td>
<td>1.94</td>
<td>1.79</td>
</tr>
<tr>
<td>2000000</td>
<td>2.37</td>
<td>2.22</td>
<td>2.09</td>
</tr>
<tr>
<td>3000000</td>
<td>2.57</td>
<td>2.14</td>
<td>2.06</td>
</tr>
<tr>
<td>4000000</td>
<td>2.55</td>
<td>2.03</td>
<td>2.14</td>
</tr>
<tr>
<td>5000000</td>
<td>2.88</td>
<td>2.28</td>
<td>2.38</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Pthread</th>
<th>Tasks</th>
<th>Parfor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>2.62</td>
<td>1.94</td>
<td>2.27</td>
</tr>
<tr>
<td>2000000</td>
<td>3.11</td>
<td>2.14</td>
<td>2.48</td>
</tr>
<tr>
<td>3000000</td>
<td>3.25</td>
<td>2.17</td>
<td>2.52</td>
</tr>
<tr>
<td>4000000</td>
<td>3.17</td>
<td>2.13</td>
<td>2.41</td>
</tr>
<tr>
<td>5000000</td>
<td>3.48</td>
<td>2.22</td>
<td>2.75</td>
</tr>
</tbody>
</table>
Introduction

CPU-cores load with **Pthread**

CPU-cores load with **Tasks**

CPU-cores load with **Parfor**

---

**Seamless Parallelism in MATLAB** by Claude TADONKI & Pierre-Louis CARUANA

Parallel and Distributed Computing and Networks (PDCN 2014) – *Feb 17-19, Innsbruck (AUSTRIA)*
THANKS FOR YOUR ATTENTION!

QUESTIONS?