[QI;MP]

MATLAB & Computing @ OIST

Lee James O'Riordan

OIST

Cluster

Computer architecture

- Looping over data with single instruction (SISD)
- Perform same single instruction over all data (SIMD)
- MIMD: multiple instruction, multiple data. Many different operations over many processors.
    - SPMD: Single program, multiple data. Most common form of parallelism.

Parallel Computing Toolbox

- MATLAB can be parallelised (provided the PCT is available --- we have it!).
- 'Local' parallelism, work split over all cores.
  - Can give reasonable speed increases.
- 'Distributed' parallelism, work split over many machines (e.g. compute cluster)
  - Great for big jobs that would otherwise take days/weeks/months/years!

Facilities at OIST

- OIST has a cluster (Tombo) with ~4000 cores.
  - Each node has 2x 6 core processors, and 48GB RAM.
  - Can we run MATLAB over all these cores? (YES!)
  - Best not to take all the cores though (other people need to use them too!).
- Setup instructions @ https://groups.oist.jp/hpc/
- 8x Nvidia Tesla M2090 GPU's (mine!)

Parallel computing with MATLAB

- spmd: offers the highest amount of control over parallelisation.
    - Should give the highest attainable performance is used appropriately.
    - Can also be needlessly complicated at times.
- parfor (parallel for): essentially a for loop that runs as parallel jobs instead of sequential looping.
    - Not as flexible or fast as spmd, but simpler is often better!

Example

- Begin by telling MATLAB to acquire N-processors

```
>> matlabpool open N %Only if N processors are free
```

- Traditional MATLAB for loop and ugly expression:

```
>> tic;
     for ii=1:100000
     exp(-ii.^2)*gamma(log(ii))*cos(mod(ii,2*pi));
     end;
   toc;
```

- Parallel for loop and ugly expression:

```
>> tic;
     parfor ii=1:100000
     exp(-ii.^2)*gamma(log(ii))*cos(mod(ii,2*pi));
     end;
   toc;
```

Example

- As before, tell MATLAB to acquire N-processors

```
>> matlabpool open N %Only if N processors are free
```

- Time the creation of a 32768x32768 random matrix, and calculate the square of each element.
- Now, to do this in parallel, need a matrix distributed over cluster nodes:

```
>> tic;
     spmd
            a=codistributed.rand(32768,32768);
            a.^2;
     end
   toc;
```

Example

- Parallelisation can be tricky!
  - Easy to do for embarrassingly parallel problems
  - Some problems require much more thought
- Consider previous example, but with matrix-matrix multiplication

```
>> tic;
     spmd
             a=codistributed.rand(32768,32768);
             a^2;
     end
   toc;
```