Lecture 3: Parallel computing with MATLAB

Examples on optimization problems

Heikki Apiola

April 16, 2019

Aalto University
heikki.apiola@aalto.fi,juha.kuortti@aalto.fi

Parallel Computing Toolbox

General concepts

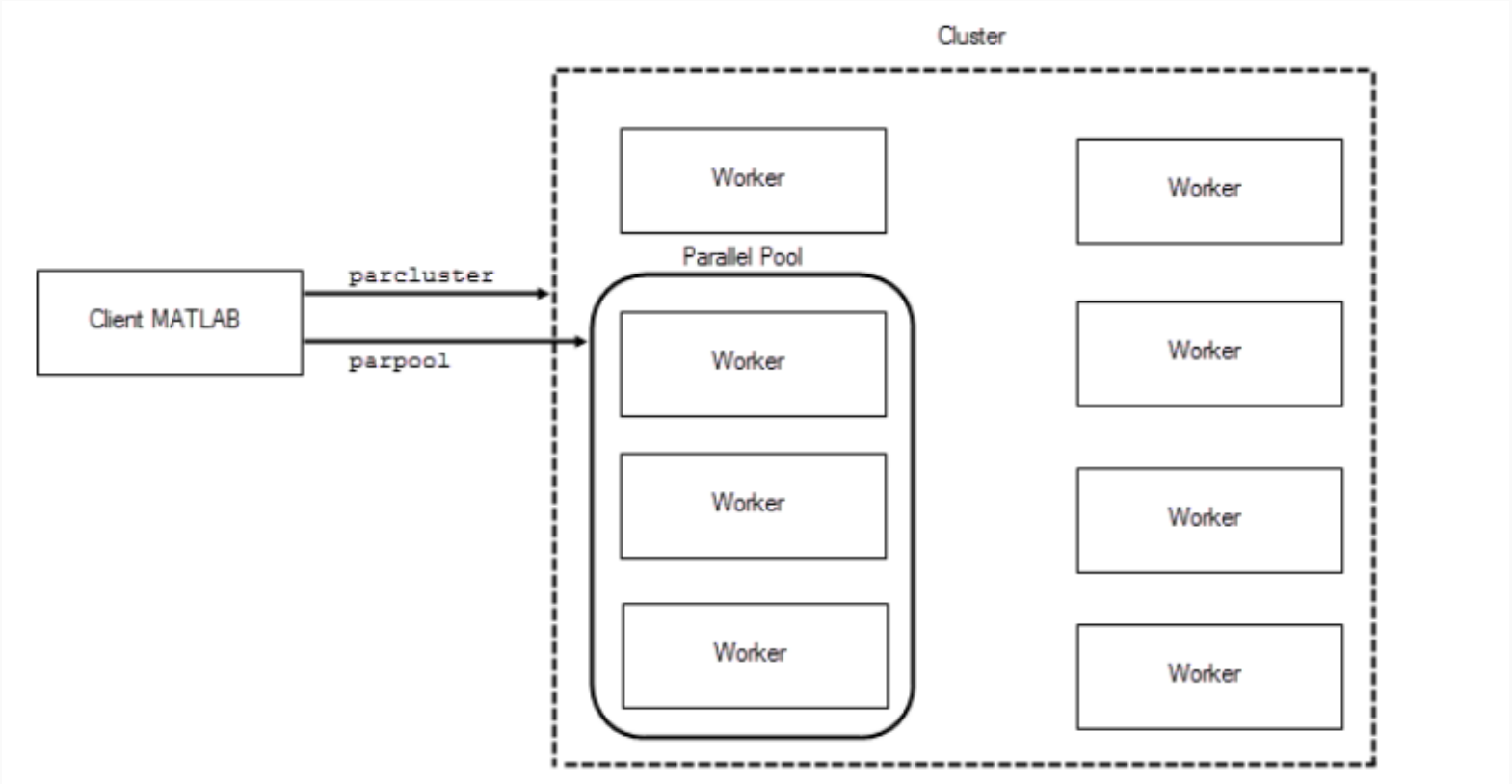
The PCT (Parallel Computing Toolbox) allows exploitation of multicore processors, and it supports the use of computer clusters and graphics processing units (GPUs).

Goals of parallel computing

- **Efficiency**: Gain remarkable speedup distributing a computational task between several computing units, which work simultaneously with parts of the task. (Decompose your task into independent parts that may or may not communicate with each other.)
- **Memory**: Solve larger problems than can be done in one computational unit.

Parallel pool

A parallel pool is a set of MATLAB® workers on a compute cluster or desktop.



More details: MathWorks:Run Code on Parallel Pools

Difficulties with parallel computing

- Computer architectures evolve $\Rightarrow$ programming techniques have to evolve as well.
- How the data has to be stored and shared between computational resources
- In some cases it may be difficult to achive satisfactory speed improvements, especially when there's need to communicate between different computational units.
- Writing and debugging parallel code is often difficult

What Matlab's PCT is good for

- Relatively easy to write parallel programs using just minor extensions of the language.
- **Limitations:** Special parallel computing languages provide more flexibility (but require more programming effort).
- Prototype parallel code by working on your local (at least two kernel) computer before running on a remote cluster.

Note: The cluster on which you run jobs must be running the `MATLAB` Distributed Computing Server, such as our *Triton*.

Parallel pool, workers, client

Terminology:

- *Client*: Matlab-session running on your laptop or remote computer (like Triton).
- *Workers* are MATLAB computational engine processes running on laptop or remote cluster. Workers don't have a MATLAB desktop, they can communicate between themselves and the client.
- *Pool*: A set of workers forms a parallel pool. A pool can be started with the command `>>parpool` and closed with `>>delete(gcp)`

Example: Open pool, doc parallel.Pool

```
>> p=parpool
Starting parallel pool (parpool) using the 'local' ...
    profile ...
connected to 2 workers.
p =
 Pool with properties:
           Connected: true
           NumWorkers: 2
              Cluster: local
        AttachedFiles: {}
    AutoAddClientPath: true
          IdleTimeout: 30 minutes (30 minutes remaining)
          SpmdEnabled: true
>> p.NumWorkers
ans =
     2
```

Controlling pool, profile

- `parpool` - Open parallel pool with default profile.
- `parpool(n)` - Open parallel pool with n workers.
- `delete(gcp)` - Close pool (default idle time: 30 min).

Most of the parallel computing constructs (`parfor`, `spmd`,etc.) to be discussed, will automatically open a pool with default profile.

Note: The parallel profile can also be viewed and edited by opening the "Parallel"-tab at the bottom right part of the "home"-view of the desktop. For more, see the Mathworks' pages which lead us to our next topic:

https://se.mathworks.com/help/distcomp/spmd.html and https://se.mathworks.com/help/distcomp/run-code-on-parallel-pools.html

Parallel computing constructs

spmd,parfor,...

We will be concerned with

- `spmd` : Single program multiple data
- `parfor` : parallel for-loop
- `parfeval` (possibly) : parallel evaluation

The `spmd`-construct gives the user full control and understanding of the parallelization process. It allows the same code to be run on multiple workers with each worker using different data, which might for example be different parts of the same array. `parfor` is technically simplest, just replace `for` by `parfor`. More on this! MATLAB decides how to distribute the computation among the workers. Some limitations compared to `for`: nested loops are not allowed and some loop index limitations exist, etc.

Toolbox specific parallelization tools

- Some toolboxes, like the (Global) optimization toolbox provide support for parallel computation.
- For example there is a setting `'UseParallel',true` in the `optimoptions` structure passed to the solver. We will look at these more closely in our examples.

Single program, multiple data – spmd

The `spmd` statement defines a block of code to be run simultaneously on multiple workers that should be reserved using `parpool`.

The first couple of slides contain quotations of the text in:

https://se.mathworks.com/help/distcomp/spmd.html

The general form of an spmd (single program, multiple data) statement is:

```
spmd
    <statements>
end
```

MATLAB executes the spmd body *statements* on `nlabs` workers simultaneously.

Open pool, `parpool`, `labindex`, `myPool.NumWorkers`

- Open a pool of MATLAB workers using **parpool** or have your parallel prefences allow the automatic start of a pool.
- Inside the body of the spmd statement, each MATLAB worker has a unique value of **labindex**.
- Opening the pool by `myPool=parpool` uses the default number of workers stored in `myPool.NumWorkers`.
  Alternatively one can start by
  `nlabs=16; parpool(nlabs)`
  in case there are `nlabs` workers available in the pool.
- Communication functions such as `labSend` and `labReceive` can transfer data between the workers.

Examples of spmd

```
nlabs=4;        % Or 2 on a 2-kernel laptop
% nlabs=16;     % Triton (default: 20, may vary)
parpool(nlabs) % Opening the pool takes some time.
spmd
  % build magic squares in parallel
  q = magic(labindex + 2);
  end
  for ii=1:length(q);figure,imagesc(q{ii});end
  % q is ``alive'' even outside the spmd-block.
```

- The variable `q` is a Composite object. The value `q{k}` is the value stored in the k^{th} worker, whereas `q(k)` is the cell array of values stored in the k^{th} worker.
- The indexing of a composite object is exactly similar to that of a *cell array*.

Composite objects

Note: As the last line showed us, the **composite object** `q` remains available even outside the `spmd`-block as long as the pool is open.

```
>> q
q =
   Lab 1: class = double, size = [3  3]
   Lab 2: class = double, size = [4  4]
>> q{1:nlabs}
ans =
     8     1     6
     3     5     7
     4     9     2
ans =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Composite objects to cell arrays, results from workers to client

As noted above, the variable `q` "behaves" like a cell array. Here's one way of copying the contents of q from workers to the client:

```
Q=cell(1,nlabs);          % Create a cell array Q.
Q(1:nlabs)=q(1:nlabs);    % Copy q into cell array Q.
% Q=q;                    % Not allowed.
delete(gcp)    % q no more available, Q remains.
cellplot(Q), figure
surf(Q{nlabs})  % Plot (surf) the last (largest) magic.
```

Hands on spmd

Exercise:Experiment spmd basics in MATLAB.

Rules on pools and spmd blocks

- If you started a pool for nlabs=20 (eg.), you have to close it before changing to parpool(4) (eg.)
- If you have a pool open, you can do several spmd-blocks one after another. So the end statement of your spmd-block doesn't close the pool. Also the composite objects remain as long as the pool is open.

- Do that
- spmdex1.m
- spmdExamplesLIVE.pdf
- spmdExamples.m
- LH

Run the same script in Triton

```
slogin -X -lscip triton.aalto.fi
Mat...xLab...y2018
$ cd $WRKDIR
$ mkdir USER_OWN_DIR
module load matlab
```

```
>>parpool
  Pool with properties:
            Connected: true
            NumWorkers: 20
            Cluster: local
            ...
>>spmd
>> q = magic(labindex + 2);
>>end
```

Continued ...

```
>> q(1:6:24)
ans =
  1×4 cell array
   {3×3 double}{9×9 double}{15×15 double}{21×21 double}
>> surf(q{24})
```

Note: The "prototype code" (on laptop) and the remote computer code (on Triton) needs no change, the latter, using Triton default uses 20 workers.

```
>> Q=cell(1,20);
>> Q(1:24)=q(1:20);
>> delete(gcp)    % close pool
                  % q is gone, Q remains.
```

Spmd-example: Numerical integration

Task: Integrate numerically: $\int_0^1 \frac{4}{1+x^2} dx$,
decomposing the interval of integration "Labwise".

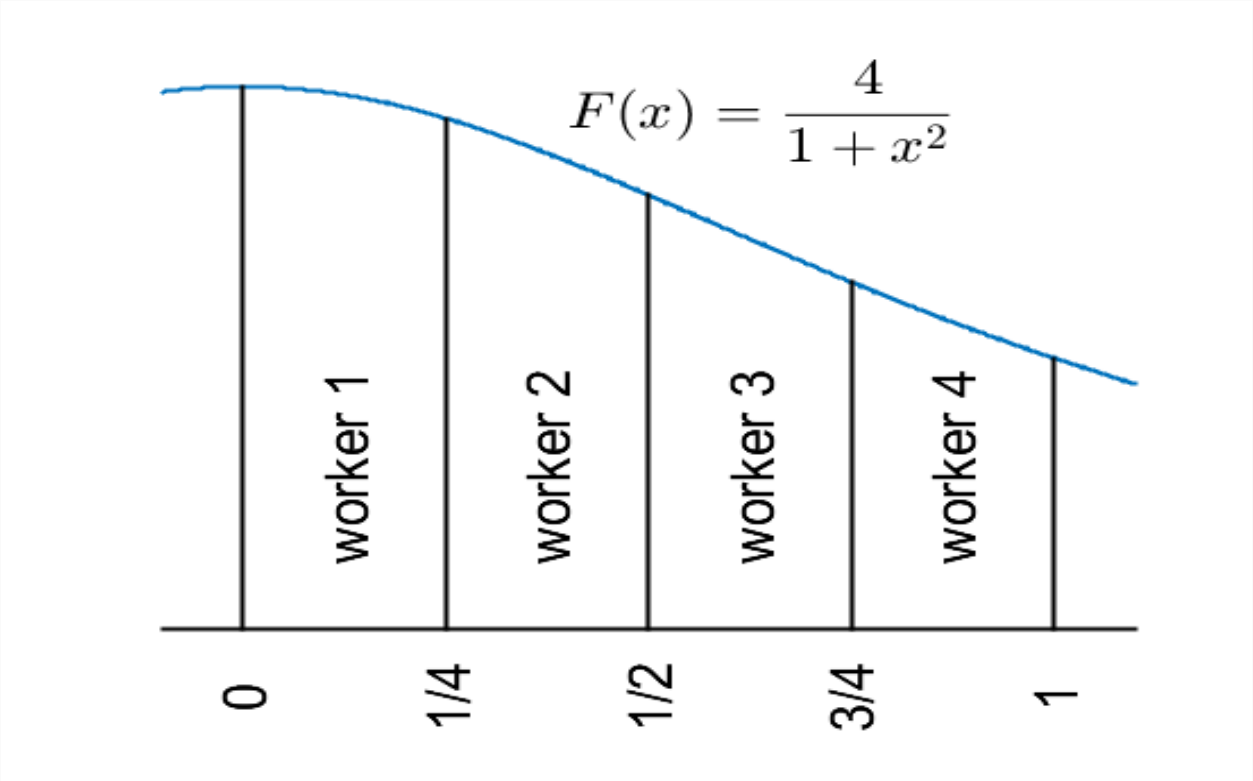


Figure from MatkWork's documents

Define the variables a and b on all the workers, but let their values depend on `labindex` so that the intervals $[a, b]$ correspond to the subintervals shown in the figure.

Note: The code in the body of the `spmd` statement is executed in parallel on all the workers in the parallel pool.

```
p=parpool  % If not open. (Takes a while.)
numlabs=p.NumWorkers
f=@(x) 4./(1+x.^2)
spmd
    a = (labindex - 1)/numlabs;
    b = labindex/numlabs;
    %fprintf('Subinterval: [%-4g, %-4g]\n', a, b);
    Int=integral(f,a,b)
    S=gop(@plus,Int)
end
```

More examples on spmd

- Find all local min/max and/or zeros of a univariate function. Exercise on this.
- Multivariate optimization, "domain decomposition"
- "Method of lines" for certain PDE's, semidiscretization, uses a large number of calls to ODE-solvers. (Matlab-demo exists.)
- "Parameter sweep" on differential equations, optimization and many others. (One example on diff, equ. included as a link.)

Some of the above may be well (or better) suited to `parfor`

Parallel for – parfor

The easiest way to use the parallel toolbox is to replace `for` by `parfor`. A `parfor`-loop splits the computation inside the loop among available workers in an automatic way, in an unspecified order.

Restrictions:

1. The loop variable must increase in steps 1.
2. No data dependencies between different iterations. Example:

```
s=ones(10,1);
parfor i=1:6,s(i)=i^2,end                    % Allowed
parfor i=2:6, s(i)=s(i-1)^2+i^2,end          % Not ...
    allowed (try)
```

3. A `parfor`-loop can't contain another for-loop, but it can contain a call to a function that contains a parfor loop