



LUND
UNIVERSITY

MATLAB in HPC

Anders Sjöström

LUNARC

anders.sjostrom@lunarc.lu.se



Why use MATLAB for HPC?

- Familiar environment
- Portability
- Rapid prototyping
- License conservation
- Computational speed
- Lots of available packages and code



MATLAB@HPC2N

- Available on the Desktop and through terminal
- Integrated with the queueing system
- Distributes work on workers in a parallel-pool
- Up to 500 workers
- Larger memory and more cores
- Can use GPU if available
- Arrays can be distributed over all workers
- Can run applications Interactively or as Batch jobs



Desktop or Console?

- For full MATLAB interface the desktop is recommended
- If only the ability to send in scripts is needed, the console can suffice
- The Desktop is run through the ThinLinc client (available at www.cendio.com/thinlinc/download)



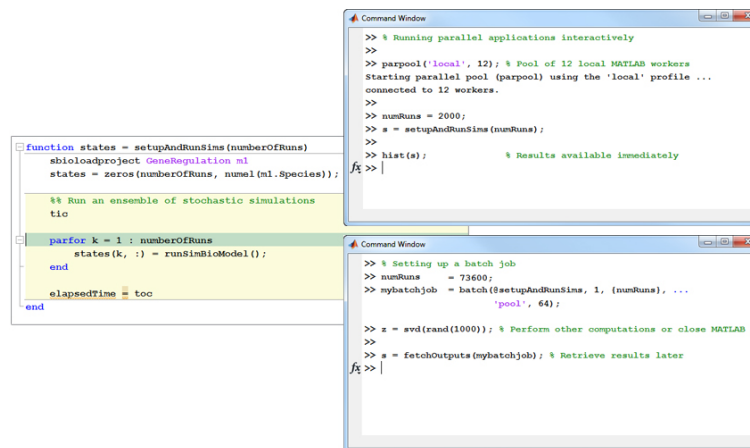
Slow code? Out of memory?

- Know your enemy:
- Understand the constraints
- Identify bottlenecks
- Exploit data and task parallelism
- Find the best trade-off between programming effort and achieving your goals



Slow code? Out of memory?

- Test your code locally on a small problem first
- Use MATLAB profiler to optimize code and spot parallelizable regions
- Avoid running interactively to the backend, use Batch instead
- Use MATLAB on the desktop as a launcher through the Batch-command



The image displays MATLAB code and command window outputs. On the left, a function `setupAndRunSims` is shown, which uses `parfor` for parallel execution. On the right, two command windows show the execution of `parpool` and `batch` commands.

```
function states = setupAndRunSims(numberOfRuns)
    sbioloadproject('GeneRegulation', ml);
    states = zeros(numberOfRuns, numel(ml.Species));

    %% Run an ensemble of stochastic simulations
    tic

    parfor k = 1 : numberOfRuns
        states(k, :) = runSbiModel();
    end

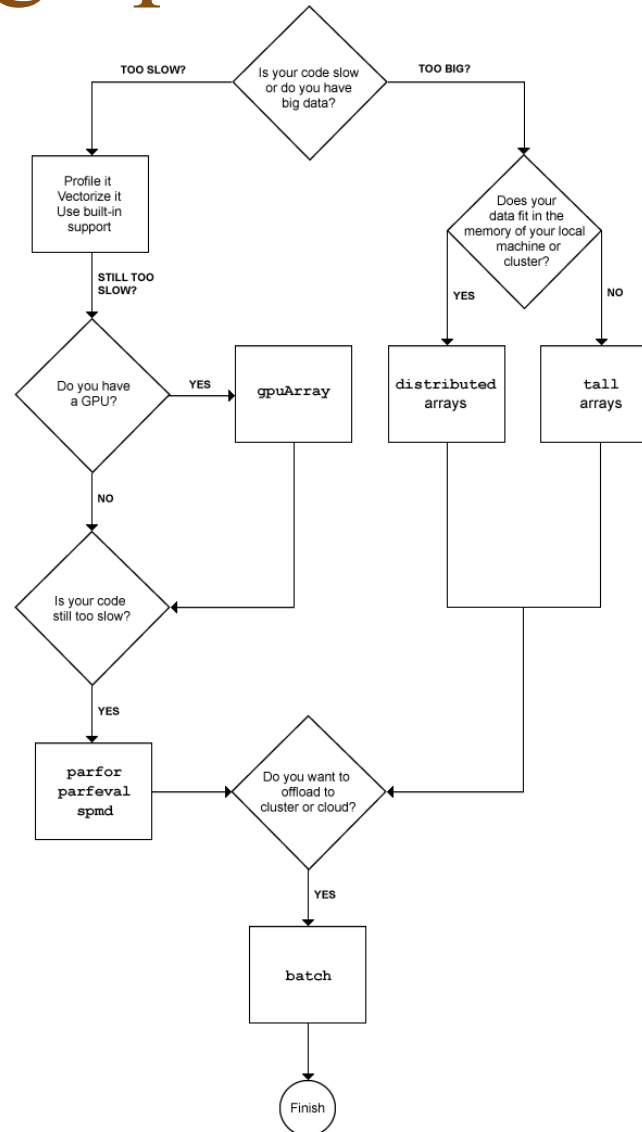
    elapsedTime = toc;
end
```

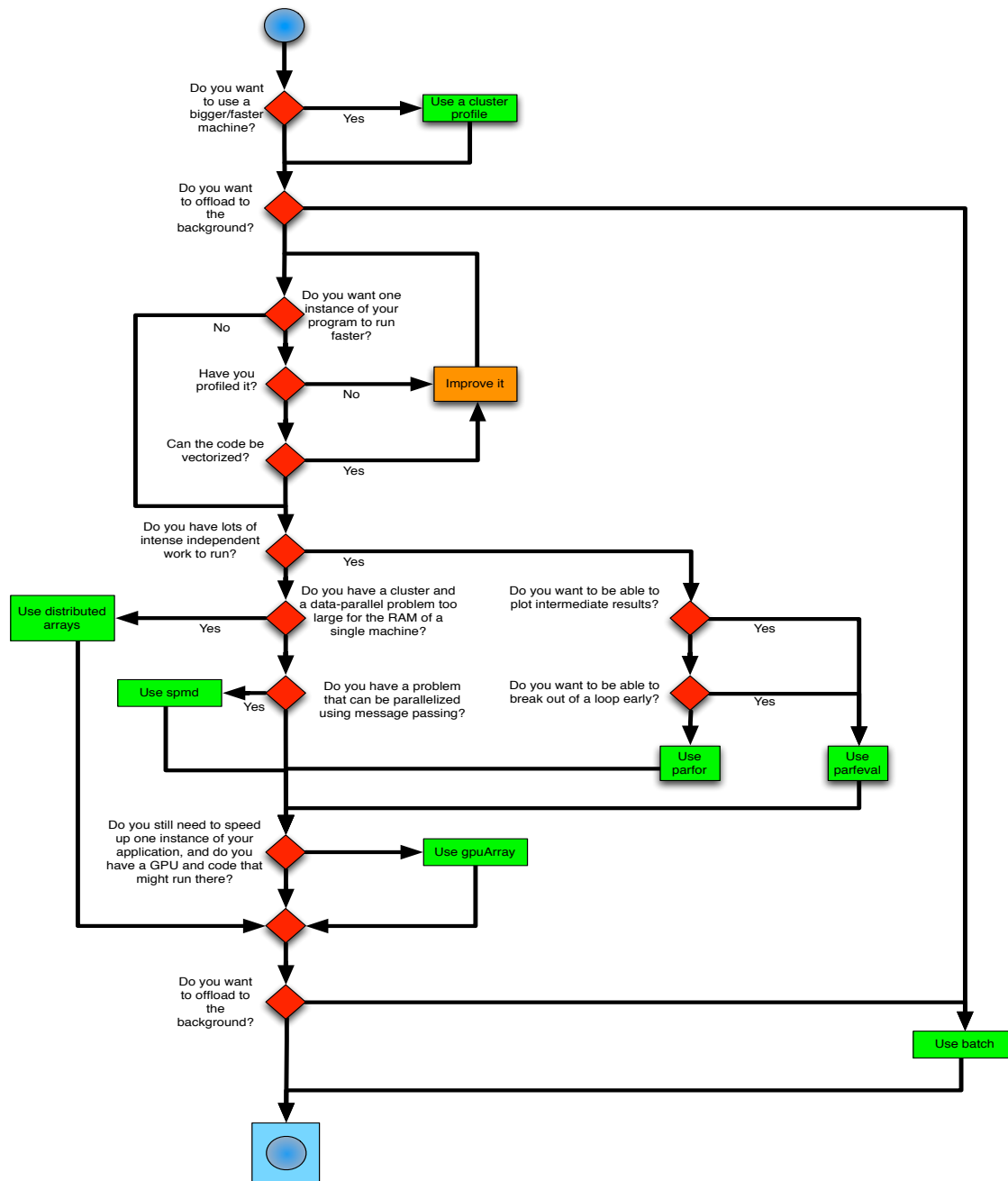
```
>> % Running parallel applications interactively
>>
>> parpool('local', 12); % Pool of 12 local MATLAB workers
Starting parallel pool (parpool) using the 'local' profile ...
connected to 12 workers.
>>
>> numRuns = 2000;
>> s = setupAndRunSims(numRuns);
>>
>> hist(s); % Results available immediately
fg>>
```

```
>> % Setting up a batch job
>> numRuns = 73600;
>> mybatchjob = batch(@setupAndRunSims, 1, (numRuns), ...
    'pool', 64);
>>
>> s = svd(rand(1000)); % Perform other computations or close MATLAB
>>
>> s = fetchOutputs(mybatchjob); % Retrieve results later
fg>>
```



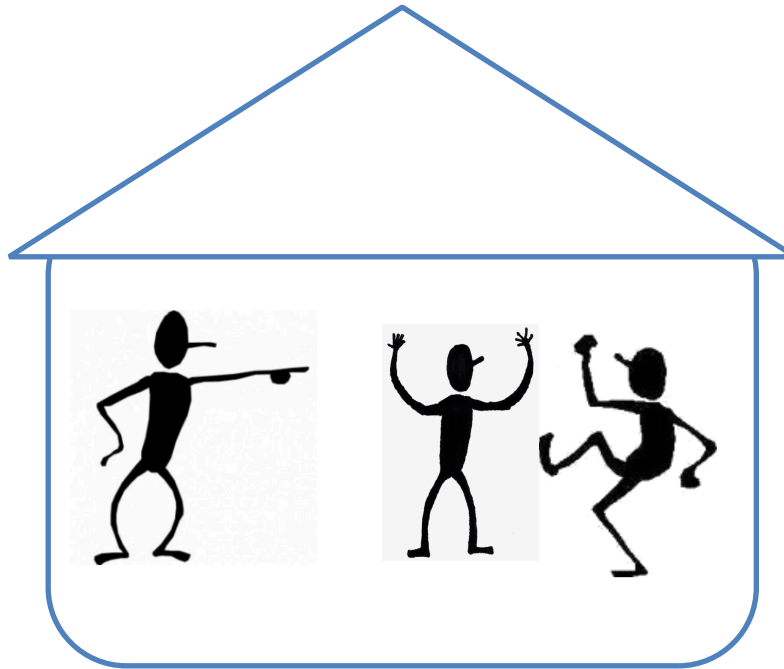
Speeding up MATLAB Flow-chart





Workflows

- Local

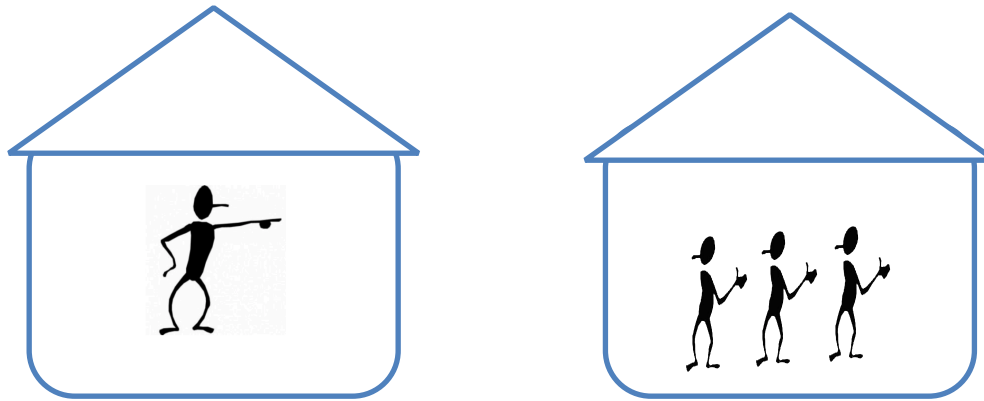


All workers are running on the local host



Workflows

- Interactive

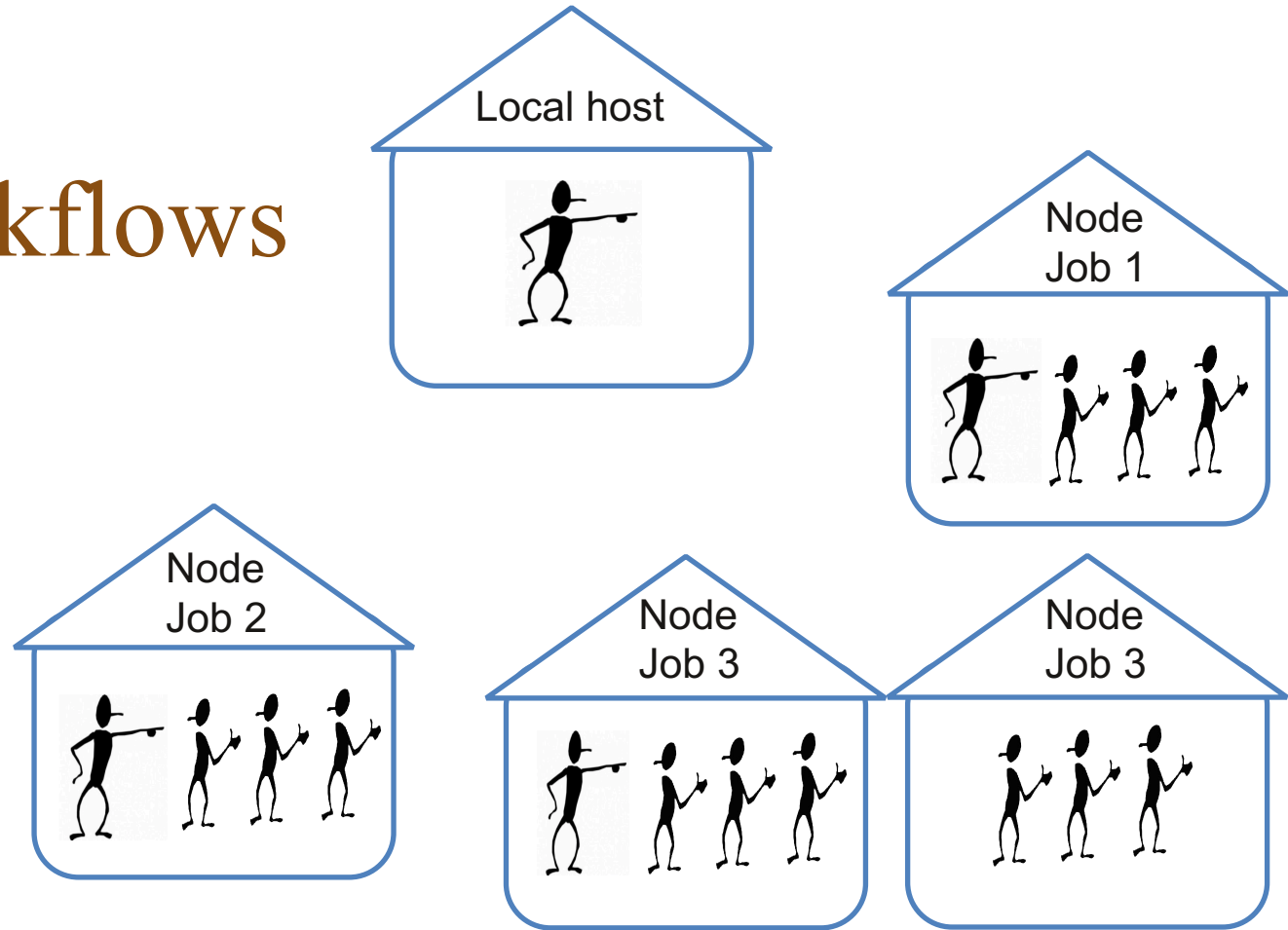


All workers are running on a remote host, controlled by the user process more or less directly



Workflows

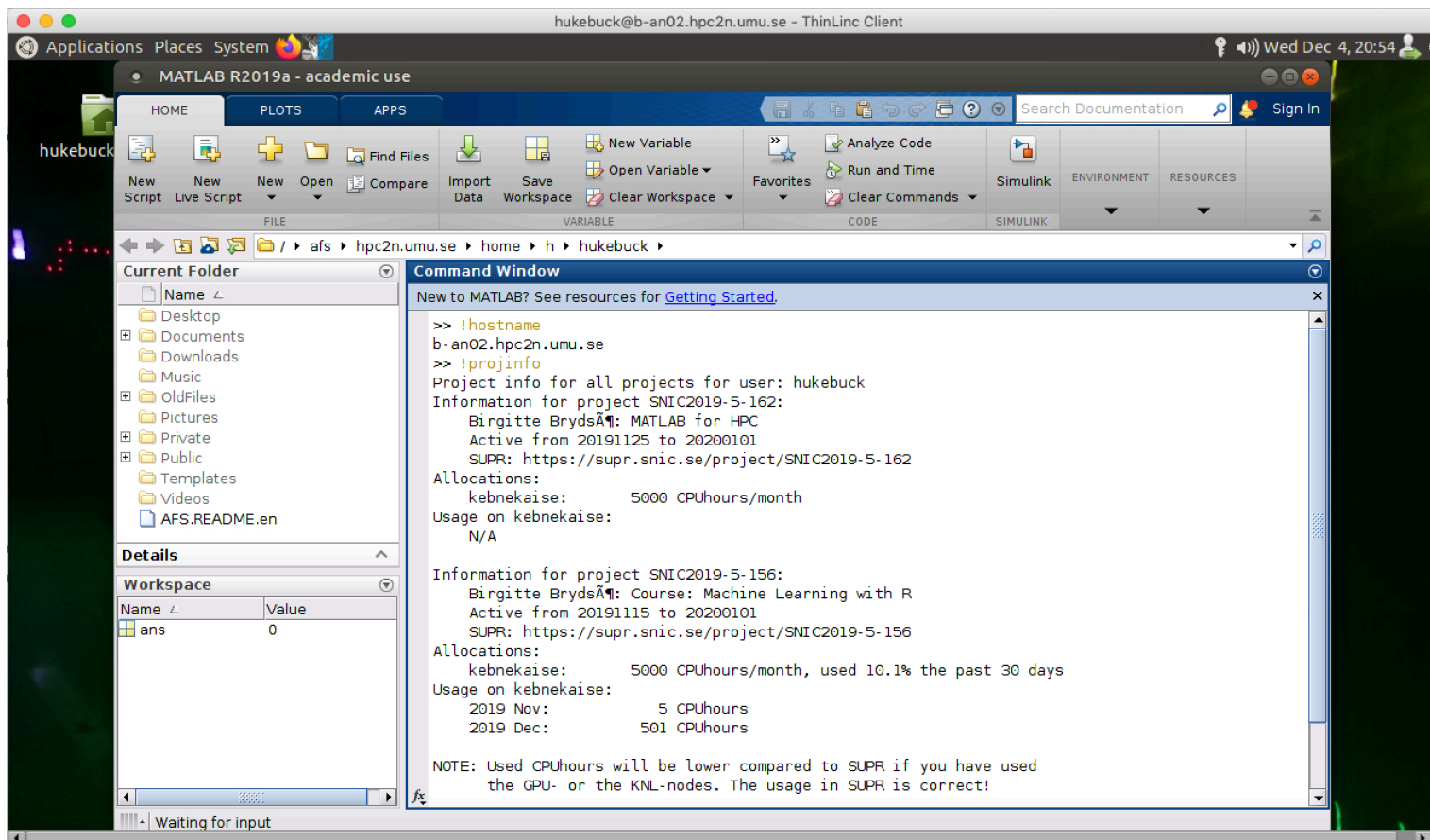
- Batch



Jobs are dispatched to workers on one or more nodes, each with its own job id in SLURM. Note that each job has its own master worker



What do I need to start a pool?



The image shows a MATLAB R2019a - academic use interface. The Command Window displays the following output:

```
>> !hostname
b-an02.hpc2n.umu.se
>> !projinfo
Project info for all projects for user: hukebuck
Information for project SNIC2019-5-162:
  Birgitte BrydsÅ: MATLAB for HPC
  Active from 20191125 to 20200101
  SUPR: https://supr.snic.se/project/SNIC2019-5-162
Allocations:
  kebnekaise:      5000 CPUhours/month
Usage on kebnekaise:
  N/A

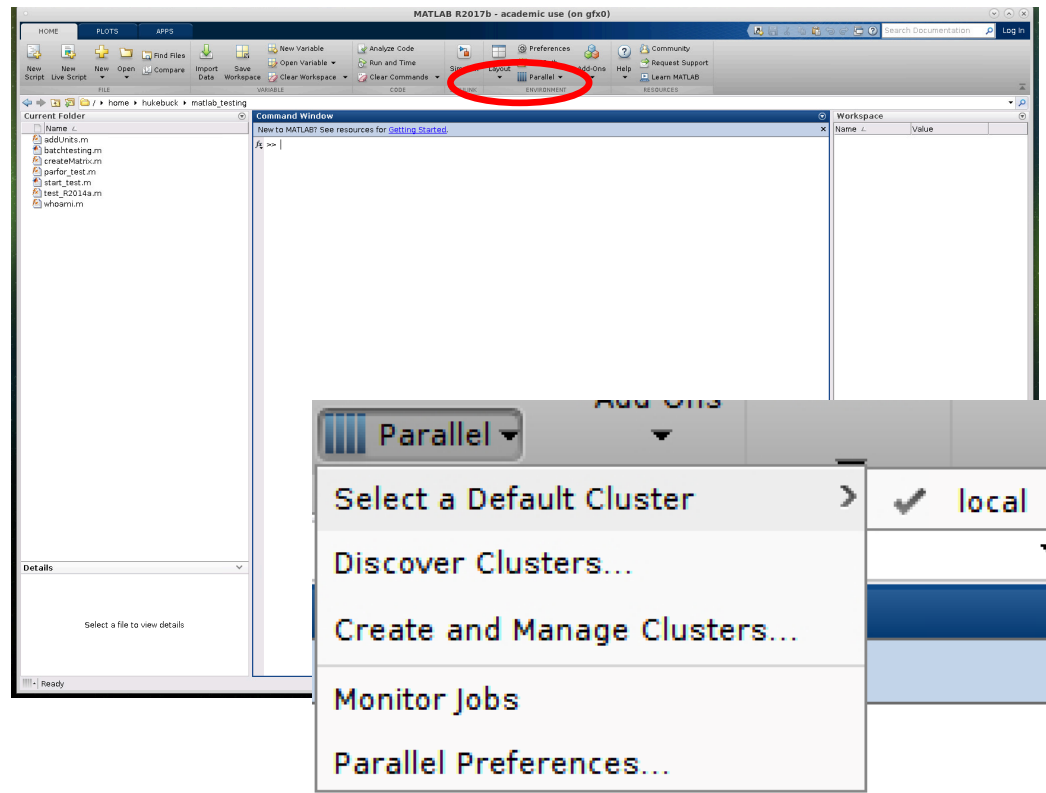
Information for project SNIC2019-5-156:
  Birgitte BrydsÅ: Course: Machine Learning with R
  Active from 20191115 to 20200101
  SUPR: https://supr.snic.se/project/SNIC2019-5-156
Allocations:
  kebnekaise:      5000 CPUhours/month, used 10.1% the past 30 days
Usage on kebnekaise:
  2019 Nov:        5 CPUhours
  2019 Dec:       501 CPUhours

NOTE: Used CPUhours will be lower compared to SUPR if you have used
the GPU- or the KNL-nodes. The usage in SUPR is correct!
```

The interface also shows a file explorer on the left with the current folder set to `afs > hpc2n.umu.se > home > h > hukebuck`. The workspace table shows a single variable `ans` with a value of `0`.



What do I need to start a pool?



What do I need to start a pool?

```
>> configCluster
      [1] abisko
      [2] kebnekaise
Select a cluster [1-2]: 2
```

Must set AccountName and WallTime before submitting jobs to KEBNEKAISE. E.g.

```
>> c = parcluster('kebnekaise');
>> c.AdditionalProperties.AccountName = 'account-name';
>> % 5 hour walltime
>> c.AdditionalProperties.WallTime = '05:00:00';
>> c.saveProfile

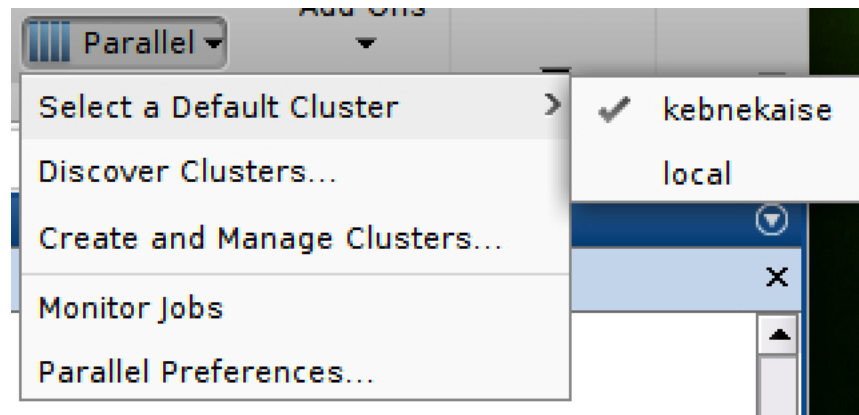
>> c=parcluster('kebnekaise');
>> c.AdditionalProperties.AccountName='SNIC2019-5-162';
>> c.AdditionalProperties.WallTime='04:00:00';
>> c.saveProfile
>> |
```

Older Installations use:

```
configCluster
ClusterInfo.setProjectName('name-of-proj')
ClusterInfo.setWallTime('00:10:00')
```



What do I need to start a pool?



What do I need to start a pool?

```
>> c.AdditionalProperties

ans =

  AdditionalProperties with properties:

    AdditionalSubmitArgs: ''
    DebugMessagesTurnedOn: 0
    EmailAddress: ''
    MemUsage: ''
    ProcsPerNode: 0
    ProjectName: 'aurora-test'
    QueueName: ''
    RequestExclusiveNode: ''
    Reservation: ''
    UseGpu: 0
    UseIdentityFile: 1
    WallTime: '00:10:00'

>>
```

Older Installations use:

```
setArch
setClusterHost
setDataParallelism
setDebugMessagesTurnedOn
setDiskSpace
setEmailAddress
setGpusPerNode
setMemUsage
setNameSpace
setPrivateKeyFile
setPrivateKeyFileHasPassPhrase
setProcsPerNode
setProjectName
setQueueName
setRequireExclusiveNode
setReservation
setSshPort
setUseGpu
setUserDefinedOptions
setUserNameOnCluster
setWallTime
```



How do I start a parallel pool?

- `parpool(n)`
 - starts a parallel pool of `n` workers
- `j = batch('script1','Pool',8)`
 - starts a pool of 8 workers and runs the script *script1* on those workers



How do I share code with workers?

- Workers Access Files Directly
- Pass Data to and from Worker Sessions
- Pass MATLAB Code for Startup and Finish



Workers access files directly

- The workers all have access to the same drives on the network, they can access the necessary files that reside on these shared resources.
- Using the job's `AdditionalPaths` property
- Putting the path command in any of the appropriate startup files for the worker

'AdditionalPaths' – A string or cell array of strings that defines paths to be added to the MATLAB search path of the workers before the script or function executes.

```
matlabroot\toolbox\local\startup.m  
matlabroot\toolbox\distcomp\user\jobStartup.m  
matlabroot\toolbox\distcomp\user\taskStartup.m
```



Pass Data to and from Worker Sessions

- **InputArguments** — Contains the input data you specified when creating the task. This data gets passed into the function when the worker performs its evaluation.
- **OutputArguments** — Property of each task contains the results of the function's evaluation.
- **JobData** — Property of the job object. Contains data that gets sent to every worker that evaluates tasks for that job. The data is passed to a worker only once per job.
- **AttachedFiles** — Property of the job object. A cell array in which you manually specify all the folders and files that get sent to the workers. On the worker, the files are installed and the entries specified in the property are added to the search path of the worker session.
- **AutoAttachFiles** — Property of the job object. A logical value to specify that you want MATLAB to perform an analysis on the task functions in the job and on manually attached files to determine which code files are necessary for the workers, and to automatically send those files to the workers.

The supported code file formats for automatic attachment are MATLAB files (.m extension), P-code files (.p), and MEX-files (.mex). Note that AutoAttachFiles does not include data files for your job; use the AttachedFiles property to explicitly transfer these files to the workers.



Pass MATLAB Code for Startup and Finish

- A worker session executes its startup.m file each time it starts.
- The startup.m file can be placed in any folder on the worker's MATLAB search path.
- These files can initialize and clean up a worker session as it begins or completes evaluations of tasks for a job.



Pass MATLAB Code for Startup and Finish

- `jobStartup.m` automatically executes on a worker when the worker runs its first task of a job.
- `taskStartup.m` automatically executes on a worker each time the worker begins evaluation of a task.
- `poolStartup.m` automatically executes on a worker each time the worker is included in a newly started parallel pool.
- `taskFinish.m` automatically executes on a worker each time the worker completes evaluation of a task.

Empty versions of these files are provided in the folder: `<matlabroot>/toolbox/distcomp/user`

You can edit these files to include whatever MATLAB code you want the worker to execute at the indicated times.

Alternatively, you can create your own versions of these files and pass them to the job as part of the

AttachedFiles property, or include the path names to their locations in the AdditionalPaths property.

The worker gives precedence to the versions provided in the AttachedFiles property, then to those pointed to in the AdditionalPaths property. If any of these files are not included in these properties, the worker uses the version of the file in the `toolbox/distcomp/user` folder of the worker's MATLAB installation.





LUND
UNIVERSITY