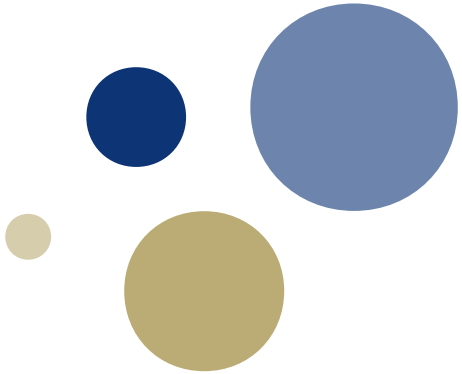
NTNU – Trondheim
Norwegian University of
Science and Technology



Introduction
to
Techila Distributed Computing Solution

John Floan NTNU-IT HPC Section
john.floan@ntnu.no
www.hpc.ntnu.no

Agenda

-What is Techila?
-Parallel loop
-Parallel region

What is Techila distributed computing solution?
The Techila is a service at NTNU ment for helping typical Matlab or Python users to speedup their code without any «major» changes in the code and without application bureaucracy.

As an user; you can run your Matlab/Python/Java/CLI/Perl/R code on your Laptop (Windows/Linux/Mac OS X), and with some changes in your code, offload the program to run on e.g. the Vilje HPC cluster in parallel on several compute nodes. You get access to the best CPU hardware at NTNU and also lot of memory.

Techila configuration

Laptop with Matlab and Techila client software

Lab PCs (Techila worker)

Techila server (NTNU)

Kongull cluster (Techila worker)

Vilje cluster (Techila worker)

How Techila works

Example codes:

Standard Matlab:

```
n = 1000;
A = rand(n,1);
B = rand(n,1);
C = zeros(n,1);

for i=1:n
      C(i) = A(i)*B(i);
end

display(C);
```

Matlab with Techila Cloudfor

```
n = 1000;
A = rand(n,1);
B = rand(n,1);
C = zeros(n,1);

cloudfor i=1:n
      C(i) = A(i) * B(i);
cloudend

display(C);
```

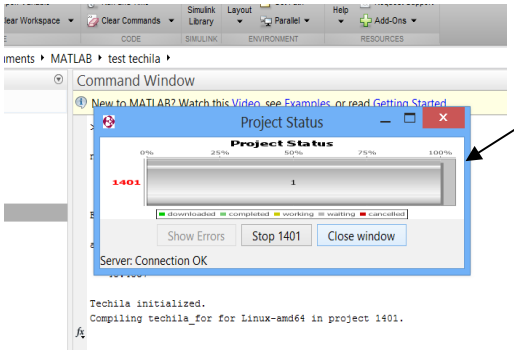
```
cloudfor i=1:n
    C(i)=A(i)+B(i);
cloudend
display (C);
```

Techila Server

Kongull

- Compute node 1
- Compute node 2
- Compute node 3
- Compute node 4

-The user start the Matlab program
-The laptop connects to Techila server when the application sees cloudfor.



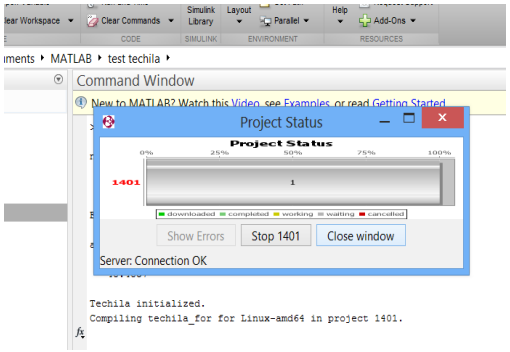
The user can see a progress bar

Techila Server

Kongull

- Compute node 1
- Compute node 2
- Compute node 3 (Compiling)
- Compute node 4

-Techila server find an available compute node for compiling (eg. node 3).



Techila Server

Kongull

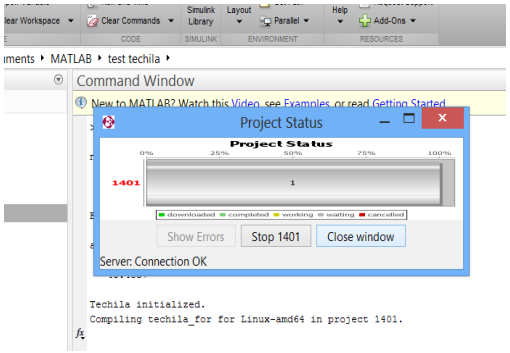
Compute node 1

Compute node 2

Compute node 3

Compute node 4

-Techila distribute the compiled code to eg.3 compute nodes (1, 2 and 4).



Techila Server

Kongull

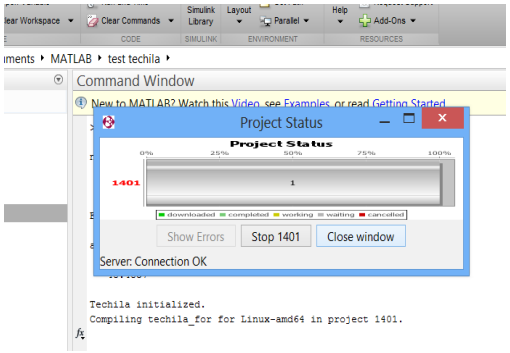
Compute node 1

Compute node 2

Compute node 3

Compute node 4

-Node 2 become occupied or dead



Techila Server

Kongull

- Compute node 1
- Compute node 2
- Compute node 3
- Compute node 4

- Techila Server move the job from node 2 to node 3

```
....
cloudend
display (C);
C =
     2    3  ....
```

Techila Server

Kongull

Compute node 1

Compute node 2

Compute node 3

Compute node 4

Techila Server collect all arrays and variables and send it to the matlab program running on the laptop.

Parallel for loop

Parallel for-loop in Matlab with Techila cloudfor:

```
%Standard Matlab     %Matlab PCT      %Techila
for i=1:n            parfor i=1:n     cloudfor i=1:n
      ....                 ....            ....
end                  end              cloudend
```

Matlab Parallel Computing Toolbox (PCT): Allows all available cores on one the computer.

Techila: Allows several computers and all available cores (in accordance with the license)

Note! Techila call CPU-core for **jobs** and also workers.

Cloudfor

Parameters:
We shall take a look at some parameters as
1 stepsperworker
2 sum
3 inputparam
4 mfilename

Parameter stepsperworker

%cf:stepsperworker : Load balancing

With stepsperworker you can control number of iterations in the for-loop per CPU core.

How to calculate the stepsperworker

Ex. You wish to use 4 CPU cores (Techila jobs) and N=250.
Stepsperworker is : 250/4 = 62.5, roundup 63.

The code:

```
nWorkers=4;
N=250;
cloudfor i=1:N
%cf:stepsperworker=63
    ....
cloudend
```

or

```
spw =  ceil ( N / nWorkers);
cloudfor i=1:N
%cf:stepsperworker=spw
   ...
cloudend
```

Example of steps per worker

You have 4 computers available with 4 cores each; totally 16 cores.

The Matlab program is as this:

```
N = 4000;
X = rand(N,1);    %One dim array
Y = zeros(N,1);

for i=1:N
   Y(i) = i * X(i)
end
```

To set the stepsperworker; do as this:

stepsperworker
= N / (4 computers * 4 cores each computer)
= 4000 / 16 = 250 iterations each core

Core 1 Core 2 ..... Core 16
for i=1:250 for i=251:500 for i=3750:4000

With Techila cloudfor:
(NB! The parameter settings are recommended)

```
N = 4000;
X = rand(N,1);    %One dim array
Y = zeros(N,1);

cloudfor i=1:N
%cf:stepsperworker=250
%cf:force:largedata
%cf:peach LocalCompile=false
%cf:peach RemoteCompile=true
%cf:peach CompilationPlatforms={{'Windows', 'amd64','mcc_64'}}
   Y(i) = i * X(i)
cloudend
...

Parameters:
%cf:force:largedata means that you can use large data arrays(>10MB).

%cf:peach LocalCompile=false means that you turn off the local
windows compiler on your laptop/pc.

%cf:peach RemoteCompiler=true means that the Techila server find a
compiler available on a Techila Worker.
  'Windows': Techila Workers with MS Windows as Lab Pcs
  'Linux':   Techial Workers with Linux as Kongull and Vilje.
```

Optimal balancing
You have to balance the steps-per-worker to the runtime is optimal.

How to find optimal runtime:
Example code (standard Matlab):

```
tic;
for ii=1:N    % Runtime (sequential) Tseq=1000 sec (17 min)
        ....
end
toc;
```

1. Measure the runtime for the standard for-loop (with tic and toc).
2. Divide the runtime with number of cores, and the optimal result is around 30 sec (NOT less than 20 sec).

Formula to find optimal number of cores:

Number of cores = T sequential / 30 sec.

Example:
You have measured that the runtime for a for loop is 1000 sec.
How many CPU cores are optimal?

Number of cores = 1000sec / 30 sec ≈ 34 cores.

Note! You may not have 34 cores available

Example (see example code optimal.m)

optimal (number of Iterations, time each iteration in sec)

Outputs are the running time for a standard matlab for loop and running time for cloudfor loop

Code

```
for ii=1:N
    runfor(Tsec); % the func runfor runs in approx T sec
end
```

Test of inputs

1. optimal (1,1) N=1 and 1 sec each iteration
2. optimal (1,10) N=1 and 10 sec each iteration
3. optimal (1,30) N=1 and 30 sec each iteration
4. optimal (2,30) N=2 and 30 sec each iteration, 1 min
5. optimal (8,30) N=8 and 30 sec each iteration, 4 min (240s)

The cloudfor loop take around 30 sec for both case 4 and 5.

Parameter: sum

Sum is a reduction parameter for gathering local sum variables to the global sum variable as :

cf:peach sum=average

Example

```
n=4000;
A=rand(1,n);
average=0;

cloudfor i=1:n
%cf:stepsperworker=1000
%cf:force:largedata
%cf:peach LocalCompile=false
%cf:peach RemoteCompile=true
%cf:peach CompilationPlatforms={{'Windows', 'amd64', 'mcc_64'}}
%cf:sum=average
     if isdeployed
         average = average + A(ii);
     end

cloudend

average= average / n;
display(average);
```

Parameter: inputparam

To avoid transfering of large amount of data
you can specify only variables and array that shall
be used inside a cloudfor

Example: Transfer array X and Z, and not Y

```
N=1000
X= rand(N,N,N);
Y= rand(N,N,N);
Z= zeros (N,N,N);

cloudfor ii=1:N
    ....
    cf:inputparam=X,Z
    Z (ii,1,1) = ii * X(ii,1,1);
cloudend
```

Parameter: cf:mfilename

You can force the techila to compile the code to a specified file with cf:mfilename.

This is important if you have serveral cloudfor after each other. If you not use mfilename; the techila compile to same file and the result is the techila recompile the code every time you runs it.

How to use mfilename:
Example

```
for t=1:N
        ....
        cloudfor ii=1:N
        %cf:mfilename=tech_cloudfor1.m
                ...
        Cloudend
        ...
        cloudfor ii=1:N
        %cf:mfilename=tech_cloudfor2.m
                ...
        Cloudend
end
```

For more parameters: See
https://www.hpc.ntnu.no/display/hpc/Techila+Distributed+Computing+Solution

Techila cloudfor is not good for this example

Example: (This program will take about 4 hours)

```
x=1; y=1;
for t=2:1000

      for ii=1:M            %This loop take 5 sec
            ....
            x = x + ii * y * X ( t ) * X ( t – 1 );
      end

      for jj=1:M            %This loop take 5 sec
            ....
            y = y + jj * x * X ( t ) * X ( t - 1 );
      end
end
```

Modified with cloudfor

```
x=1; y=1;
for t=2:1000

    cloudfor ii=1:M          % This loop take about 10sec
    %cf:mfilename=tech_cloudfor1.m
        ....
        x = x + ii * y * X ( t ) * X ( t - 1 );
    cloudend

    cloudfor jj=1:M          % This loop take about 10sec
    %cf:mfilename=tech_cloudfor2.m
        ....
        y = y + jj * x * Y ( t ) * Y ( t - 1 );
    cloudend
end
```

1. In this case we have to use cloudfor in the inner loops. The outer loop can not be parallelized.

2. Using cloudfor to each inner loop give none benefite because of the running time is to small each loop.

Techila peach

Peach is nearly the same as parallel region in OpenMP.

Techila start to run a function on several CPU cores in parallel.

Example: Hello World

```
jobs=4; %Number of matlab workers (Techila jobs)
resulttxts = ...                    % return values from the func.
peach('hello_from_worker', ...      % Function call
        {'<param>'},1:jobs,...      % input arguments: numb jobs
        'RemoteCompile','true','LocalCompile','false', ...
        'CompilationPlatforms',{{'Windows', 'amd64', 'mcc_64'}});

for ii=1:jobs
    display(resulttxts{ii});
End
```

```
function returntxt=hello_from_worker(jobid)
returntxt=['Hello world from worker: ',num2str(jobid)];
end
```

Example: Functions call

```
-------------------------------------------------------------------------------
jobs=2; %Number of matlab workers (Techila jobs)
resulttxts = ...                    % return values from the func.
peach('function_calls', ... % Function call
        {'<param>',x},1:jobs,...    % input arguments, numb jobs
        'RemoteCompile','true','LocalCompile','false', ...
        'CompilationPlatforms',{{'Windows', 'amd64', 'mcc_64'}});

for ii=1:jobs
    display(resulttxts{ii});
End

-------------------------------------------------------------------------------

function y=function_calls ( jobid , x )

    switch jobid
        case 1
            y=sin(x);
        case 2
            y=cos(x);
    end_case

end
```

Pros:

- Better use of the Computer Resources at NTNU
- «Easy» to program
- Parallel computation on several nodes
- Multiplatform support (Windows/Linux/MacOS)
- Easy to manage for an administrator
- Web GUI for monitoring the job status
- Take only one matlab license; for compiling
- Do not need Matlab Parallel Computing Toolbox
- Do not interfere with other jobs on the cluster
- Support different programming language as: Matlab, C, R and Python

Cons:

- Do not have an accounting system (not yet)
- No support for interconnection between nodes (not yet).
- Not part of the PBS scheduling system (Lower priority)
- License cost.