SKILLPILLS

Matlab Skill Pill

Lecture 4: Accelerating Matlab

Matthew Edmonds

Okinawa Institute of Science and Technology
matthew.edmonds@oist.jp

October 12th, 2017

OIST

What methods can we use to speed-up Matlab?

- Vectorization
- Using the Profiler
- Matlab Parallel toolbox
- Cluster computing methods
- GPU computing

What can we do to expedite Matlab?

SKILLPILLS

For most realistic problems of interest, Matlab is inherently *slow*.

- Write better code!
- Utilise Matlab's built in features.
- Problem reduction (If possible!)
- Matlab Profiler

Vectorization (i)

As the name suggests, Matlab is geared towards manipulating data in the form of arrays (Matrices).

Suppose we wish to populate an array, we could use:

```
c=0;
my_func=zeros(1,63);
for t=0:0.1:2*pi
  c=c+1;
  my_func(1,c)=cos(t);
end
```

Let's vectorize this code!

Vectorization (ii)

```
t=0:0.1:2*pi;
my_func=cos(y);
```

This has the same functionality as the iterated example; but Matlab handles this much faster! We can use the `tic` and `toc` commands to see the speed-up

```
tic;
... expressions ...
toc;
```

Using vectorization takes Matlab 0.3ms, using iteration takes 0.7ms, over a factor of two faster!

Vectorization (iii) - Heads or tails?

If I flip a (fair) coin many times, how many times will I get a head, and how many times will I get tails?

Vectorization (iv)- Heads or tails?

Now suppose we have a data array containing a mixture of positive and negative values, and we wish to know how many are positive. We could use

```
A=2*(rand(100,100)-0.5);
A_pos=0;
for i=1:100
 for j=1:100
  if A(i,j) > 0
   A_pos=A_pos+1;
  end
 end
end
```

Which is *very* labourious!

Vectorization (v) - Heads or tails?

Again we can exploit Matlab's native vector syntax to drastically simplify this problem

```
A=2*(rand(100,100)-0.5);
A_pos=A(A > 0);
sum(A_pos)
```

Which completely avoids using iteration and selection.

Exercise - The three sided coin

Vectorize the following Matlab code

```
B=rand(1,1000);
num_R=0; num_G=0; num_B=0;

for i=1:length(B)
  if B(i) <= 1/3
     num_R = num_R + 1;
  elseif (1/3 < B(i)) & (B(i) <= 2/3)
     num_G = num_G + 1;
  elseif (B(i) > 2/3)
     num_B = num_B + 1;
  end
end

disp(strcat('Red:',num2str(num_R)));
disp(strcat('Green:',num2str(num_G)));
disp(strcat('Blue:',num2str(num_B)));
```

Answer - The three sided coin

```
B=rand(1,1000);
num_R=length(B(B<1/3));
num_G=length(B( (B>1/3) & (B<=2/3) ));
num_B=length(B(B>2/3));

disp(strcat('Red:',num2str(num_R)));
disp(strcat('Green:',num2str(num_G)));
disp(strcat('Blue:',num2str(num_B)));
```

Vectorization Summary

As well as the computational speed-up, vectorization gives added bonuses

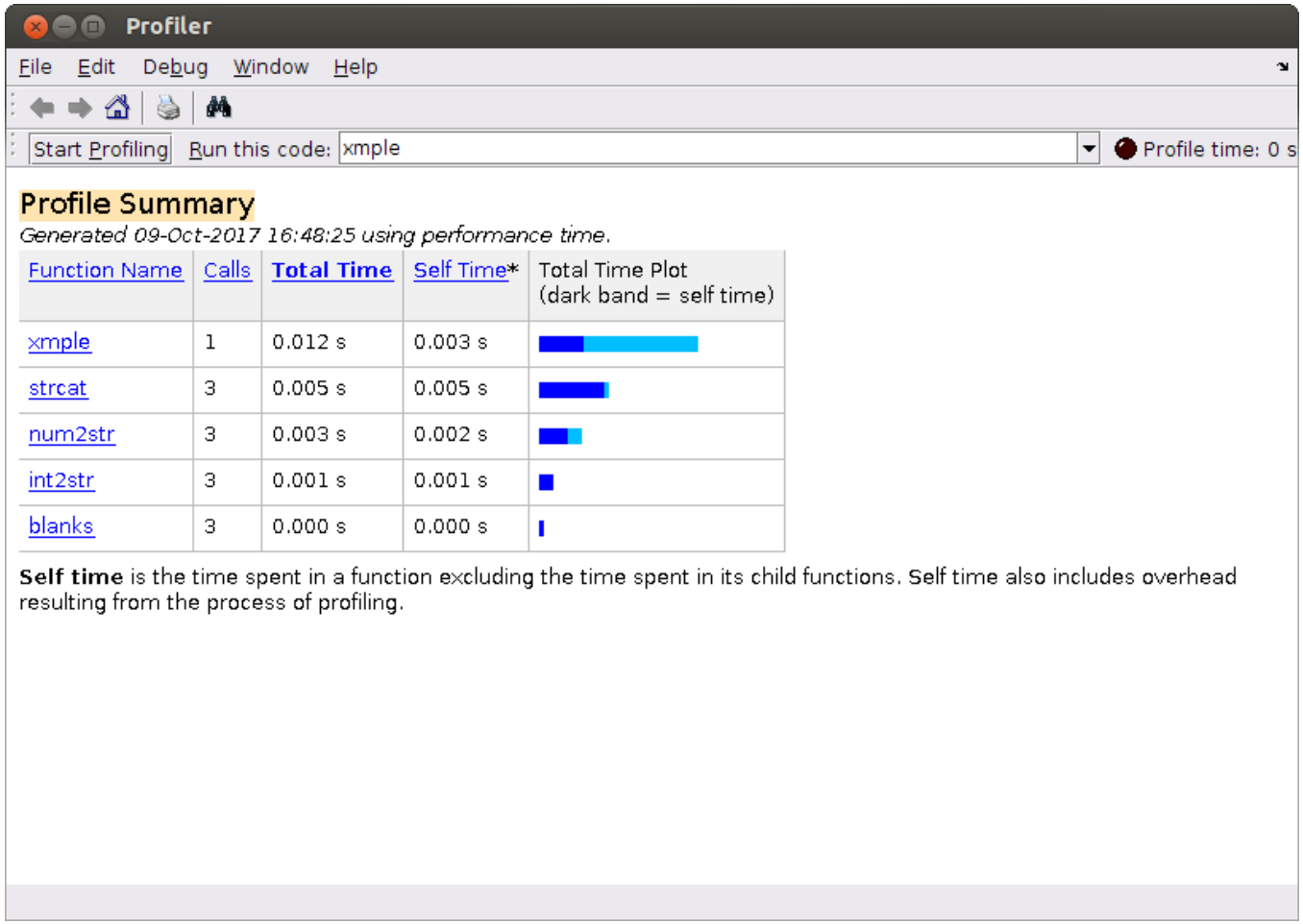
- Reduces overall code size
- Removes redundant variables
- Simpler code, less errors
- Easier to understand

Matlab Profiler

- The `tic` and `toc` commands allow us to time the total execution time of a Matlab program.
- We can use the profiler to see how long Matlab spends on each part of a program.
- type `profile viewer` to bring up the profiler

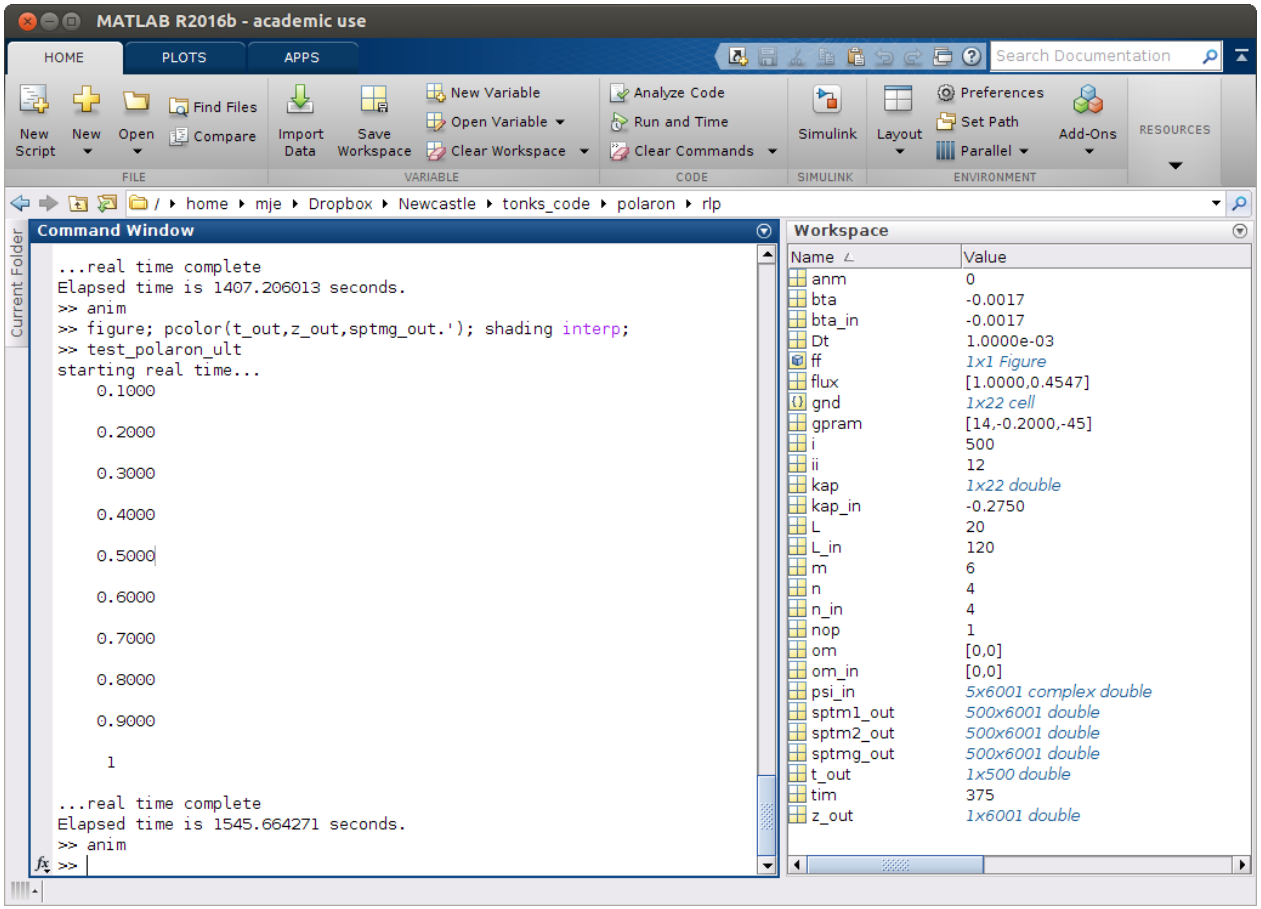
Matlab Profiler (ii)

For a given script, the profiler will generate a list of functions and the timing for each of them.



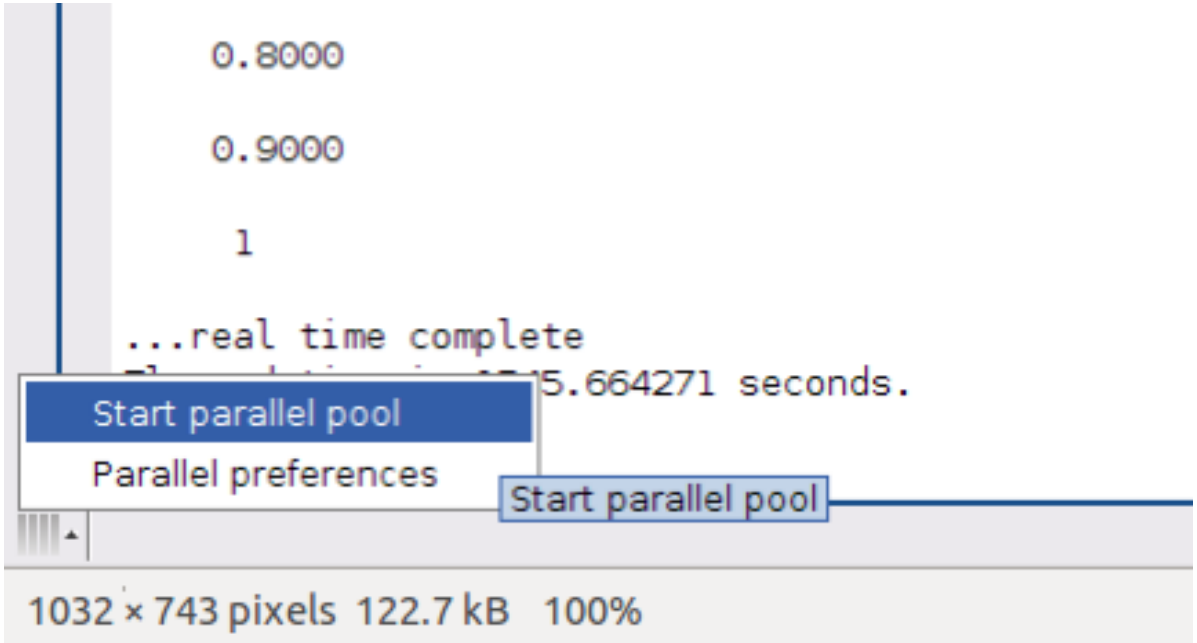
Parallel Toolbox (i)

We can take advantage of computer architecture to use multiple cores for calculations in Matlab. This is handled using the `parfor` command, which parallelises Matlab's `for` loop command.



Parallel Toolbox (ii)

We can run parallel tasks locally using the parallel toolbox, located in the bottom left of the Matlab workspace



Matlab will choose by default how many cores are in the parallel pool.

Parallel Toolbox (iii)

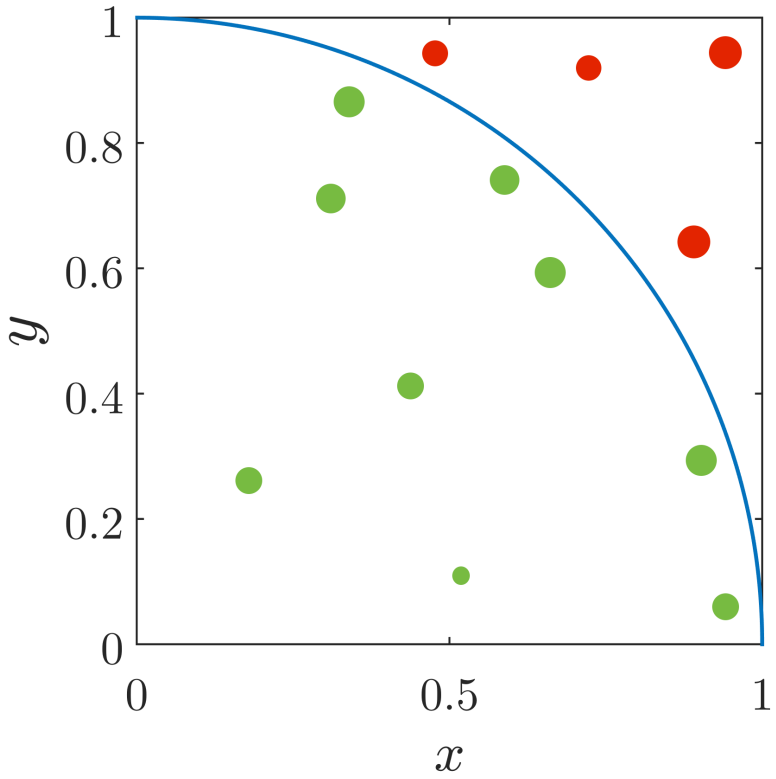
SKILLPILLS

We can also use the command `parpool(x)` to start a parallel pool with `x` cores.

```
>> parpool(3)
Starting parallel pool (parpool) using the 'local' profile ... connected to 3 workers.

ans =

 Pool with properties:

            Connected: true
           NumWorkers: 3
              Cluster: local
        AttachedFiles: {}
          IdleTimeout: 30 minute(s) (30 minutes remaining)
          SpmdEnabled: true

fx >>
```

The maximum number of cores in the pool is determined by the machine you're using.

Monte Carlo integration for π



$$A_c = \pi r^2$$

$$A_r = r^2$$

$$\frac{A_c}{A_r} = \pi$$

Parallel Toolbox (v)

```
num_samples = 1e7;
x = rand(num_samples,1,'double');
y = rand(num_samples,1,'double');
parfor ii=1:length(x)
  r(ii) = xx(ii)^2 + yy(ii)^2;
end
circle_count=0;
parfor ii=1:length(x)
  if r(ii) <= 1
    circle_count = circle_count + 1;
  end
end
mypi = 4*(circle_count/num_samples);
disp(strcat('Pi=',num2str(mypi)));
disp(strcat('Error=',num2str(100*(1-(pi/mypi))),'%'));
```

Exercise - parfor

Use vectorization and the `parfor` command to calculate, store and time the average value of four random $10^4 \times 10^4$ matrices.

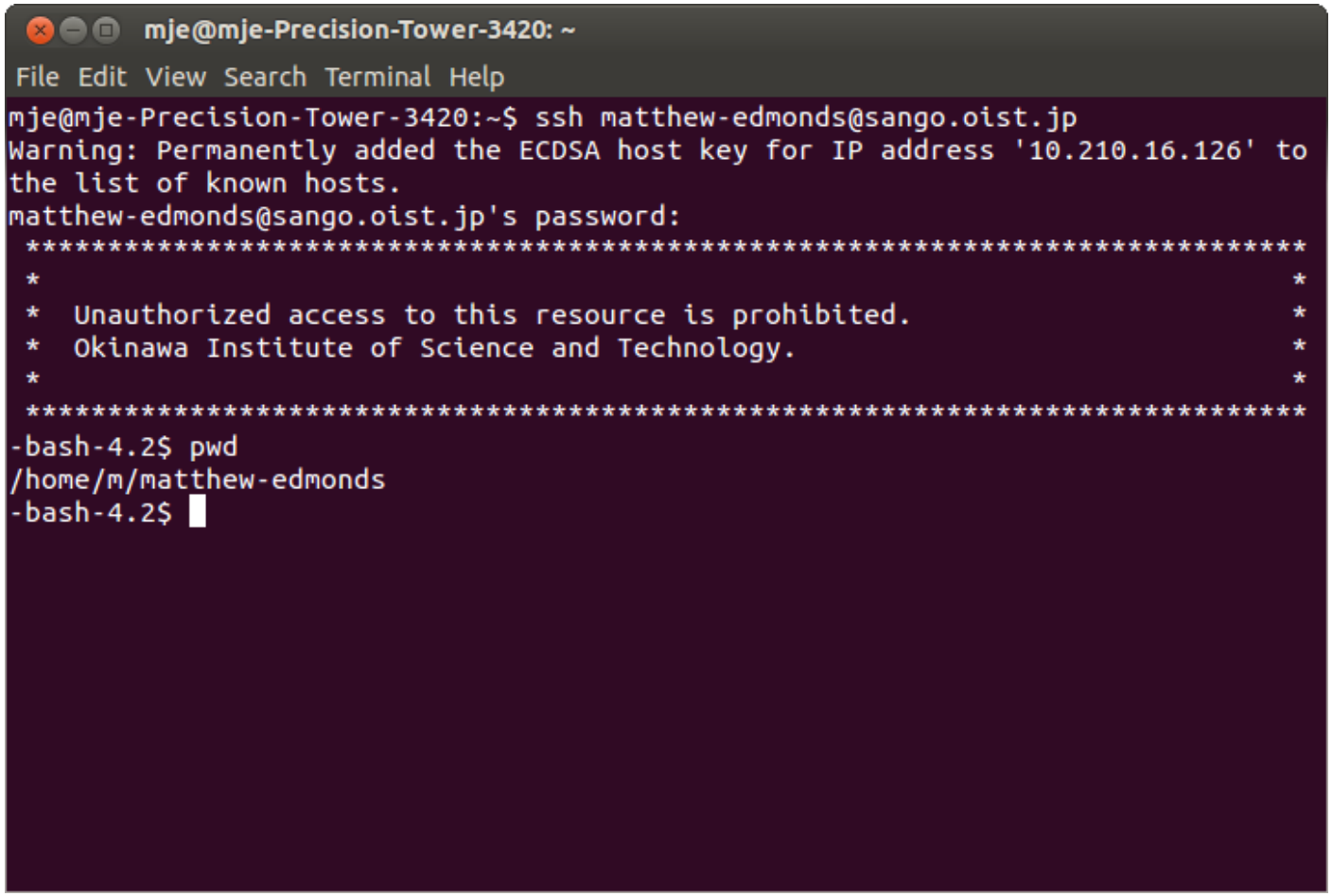
Answer - parfor

```
tic;
avg_R=zeros(1,4);
parfor i=1:4
   mat_R=rand(1e4);
   avg_R(i)=mean(mean(mat_R));
end
disp(avg_R);
toc;
```

Running on four cores this code takes $\sim$ 1.6s, without parallel $\sim$ 4s, 2.5$\times$ faster!

Cluster computing (i)

- For big calculations, can use cluster



- OIST has *Sango*, ∼10k cores!

Cluster computing (ii)

- open a terminal and execute `ssh user-name@sango.oist.jp`
- store your data in `/work/UnitName/UserName/`
- to copy a file from your machine to Sango, use:

```
scp file.m user-name@sango.oist.jp:/work/UnitName/UserName/
```

- To run a script on sango, we require a SLURM file.

Cluster computing (iii)

SKILLPILLS

- SLURM manages jobs running on Sango

```
#!/bin/bash -l

#SBATCH --job-name=JobName
#SBATCH --partition=compute
#SBATCH --mem=4G
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --time=2:00:00
#SBATCH --output=/work/UnitName/<myDir>/VOPT.log
#SBATCH --error=/work/UnitName/<myDir>/VOPT.err

cd <dir you want>
module load matlab
matlab_exe=‘matlab -nosplash -nodisplay -nodesktop’
matlab_cmd="file" #Matlab .m file without the .m

#${matlab_exe} -r "${matlab_cmd};exit" >> ./matlab.log
```

Cluster computing (iv)

- To run a job on Sango, use `sbatch job.slurm`

```
-bash-4.2$ squeue -u ${USER}
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            115228   compute     test matthew-  R       0:06      1 sango10514
```

- run `squeue -u ${USER}` to see a list of your submitted jobs, and their status.

- to cancel a job use `scancel XXXXXX`, where `XXXXXX` is your job number.

GPU computing & Matlab

Matlab calculations can also be performed on Graphics cards, using the built in GPU support.

OIST currently has nVideo Tesla K80 GPU available, by request. More information about this can be found here:

https://groups.oist.jp/scs/gpgpu-computation-using-matlab

Summary

- Vecorization is a powerful tool to both speed-up and simplify Matlab scripts.
- The profiler can be used to dissect the performance of your code.
- We can employ `parfor` to parallelise iteration.
- Cluster computing is useful for very big numerical calculations.
- GPU computing can be used for *very* big jobs!