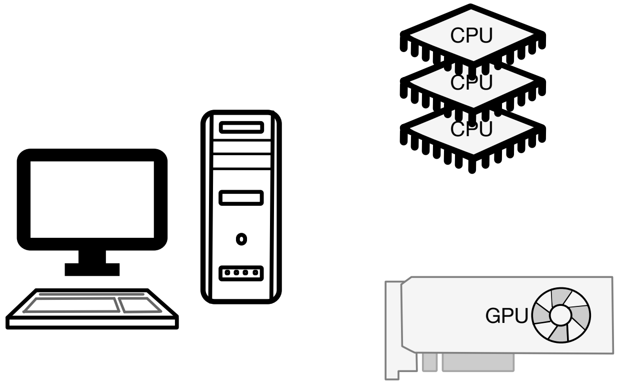
MATLAB in HPC

Parallel Computing Toolbox I.

Anders Sjöström LUNARC
anders.sjostrom@lunarc.lu.se

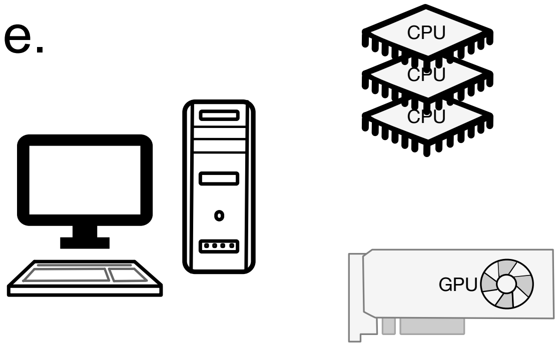
Parallel Computing Toolbox (PCT)

- Can use multicore processors, GPUs, and computer clusters.
- High-level constructs.
- No CUDA or MPI programming necessary.
- Programs and models can run in both interactive and batch modes.
- Included in standard license for single server/computer.



Parallel Computing Toolbox (PCT)

- PCT use workers (MATLAB computational engines) on multicore desktops, executing applications locally.
- With no change to the code, the same applications can be run on clusters or clouds (using MATLAB Parallel Server™).
- You can also use the toolbox with MATLAB Parallel Server to execute calculations on data too large to fit into the memory of a single machine.



Interactively Run a Loop in Parallel

Create a sine waveform and plot it

Serial version

```
for i = 1:1024
  A(i) =
sin(i*2*pi/1024);
end
plot(A)
```

Parallel version

```
parfor i = 1:1024
  A(i) =
sin(i*2*pi/1024);
end
plot(A)
```

for - parfor

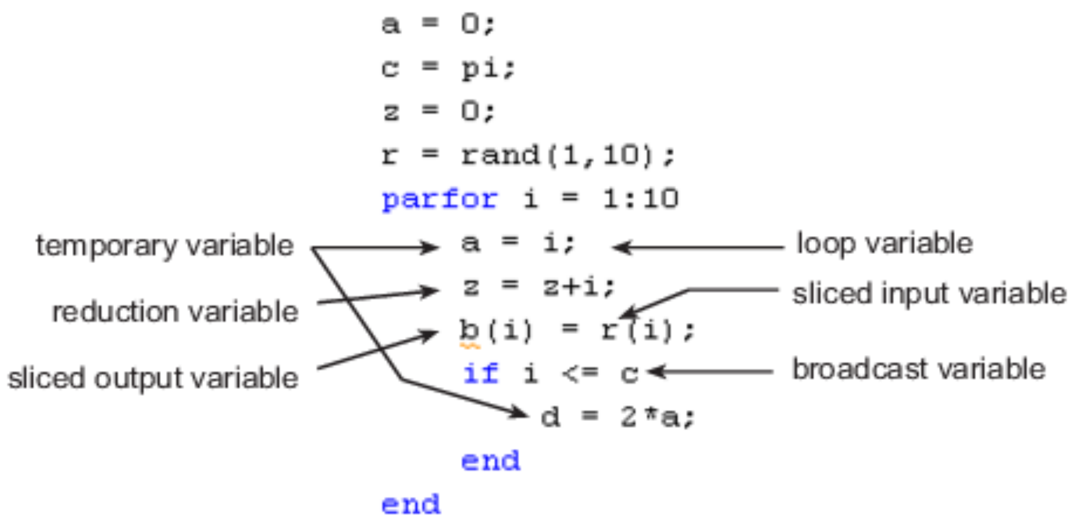
Do not need to be consecutive

```
for loopvar = initval:endval
   <statements>
   END
```

```
parfor loopvar = initval:endval
       <statements>
   END
```

Must be consecutive

Parfor variables



```
a = 0;
c = pi;
z = 0;
r = rand(1,10);
parfor i = 1:10
    a = i;
    z = z+i;
    b(i) = r(i);
    if i <= c
        d = 2*a;
    end
end
```

- Loop Variable: Loop index
- Sliced Variables: Arrays whose segments are operated on by different iterations of the loop
- Broadcast Variables: Variables defined before the loop whose value is required inside the loop, but never assigned inside the loop
- Reduction Variables: Variables that accumulate a value across iterations of the loop, regardless of iteration order
- Temporary Variables: Variables created inside the loop, and not accessed outside the loop

What if my loop is nested?

```
M1 = magic(100);
M2=zeros(size(M1));
tic;
[j,k]=size(M1);
for x = 1:j
    for y = 1:k
        M2(x,y) = x*10 + y + M1(x,y)/10000;
    end
end
t(1)=toc;
```

```
parfor x = 1:j
    for y = 1:k
        M2(x,y) = x*10 + y + M1(x,y)/10000;
    end
end
t(2)=toc;
```

0.3993 sec

```
for x = 1:j
    parfor y = 1:k
        M2(x,y) = x*10 + y + M1(x,y)/10000;
    end
end
t(3)=toc;
```

3.6440 sec

What if my loop is nested?

- The body of a parfor-loop cannot contain another parfor-loop.
- A parfor-loop can call a function that contains another parfor-loop.
- A worker cannot open a parallel pool. Thus, a worker cannot run an inner nested parfor-loop in parallel.
- Only one level of nested parfor-loops can run in parallel.
- If the outer loop runs in parallel on a parallel pool, the inner loop runs serially on each worker.
- If the outer loop runs serially in the client, the function that contains the inner loop can run the inner loop in parallel on workers in a pool.
- The body of a parfor-loop can contain for-loops.
- You can use the inner loop variable for indexing the sliced array, but only in plain form, not part of an expression.

```
A = zeros(4,5);
parfor j = 1:4
    for k = 1:5
        A(j,k) = j + k;
   end
end
A
```

What if my loop is nested?

Invalid

```
A = zeros(100, 200);
   parfor i = 1:size(A, 1)
    for j = 1:size(A, 2)
       A(i, j) = plus(i, j);
    end
end
```

Valid

```
A = zeros(100, 200);
n = size(A, 2);
   parfor i = 1:size(A,1)
    for j = 1:n
       A(i, j) = plus(i, j);
    end
end
```

For proper variable classification, the range of a for-loop nested in a parfor must be defined by constant numbers or variables. In the example, the code on the left does not work because the for-loop upper limit is defined by a function call. The code on the right works around this by defining a broadcast or constant variable outside the parfor first.

What if my loop is nested?

Invalid

```
A = zeros(4, 11);
parfor i = 1:4
    for j = 1:10
        A(i, j + 1) = i + j;
    end
end
```

Valid

```
A = zeros(4, 11);
parfor i = 1:4
    for j = 2:11
        A(i, j) = i + j - 1;
    end
end
```

The index variable for the nested for-loop must never be explicitly assigned other than in its for-statement. When using the nested for-loop variable for indexing the sliced array, you must use the variable in plain form, not as part of an expression. In the example, the code on the left does not work, but the code on the right does.

What if my loop is nested?

Invalid

```
A = zeros(4, 10);
parfor i = 1:4
    for j = 1:10
        A(i, j) = i + j;
    end
    disp(A(i, 1))
end
```

Valid

```
A = zeros(4, 10);
  parfor i = 1:4
   v = zeros(1, 10);
      for j = 1:10
        v(j) = i + j;
           end
        disp(v(1))
      A(i, :) = v;
         end
```

If you use a nested for-loop to index into a sliced array, you cannot use that array elsewhere in the parfor-loop. In the example, the code on the left does not work because A is sliced and indexed inside the nested for-loop; the code on the right works because v is assigned to A outside the nested loop.

What if my loop is nested?

Invalid

```
A = zeros(4, 10);
parfor i = 1:4
    for j = 1:5
        A(i, j) = i + j;
    end
    for k = 6:10
        A(i, k) = pi;
    end
end
```

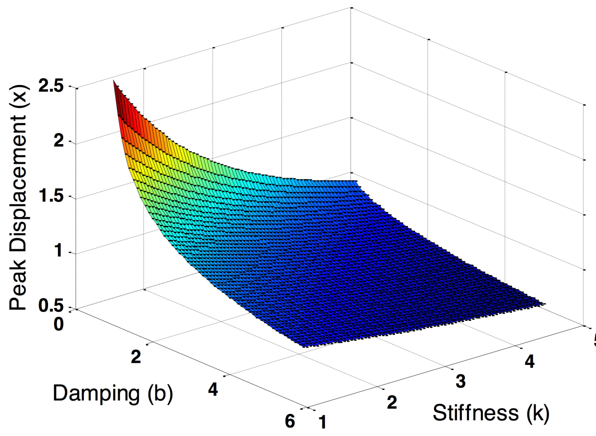
Valid

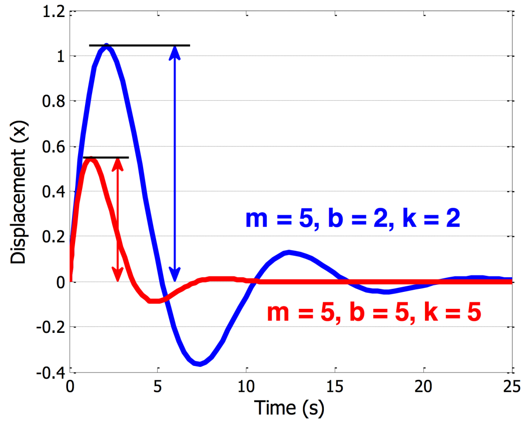
```
A = zeros(4, 10);
parfor i = 1:4
    for j = 1:10
        if j < 6
            A(i, j) = i + j;
        else
            A(i, j) = pi;
        end
    end
end
```

Inside a parfor, if you use multiple for-loops (not nested inside each other) to index into a single sliced array, they must loop over the same range of values. Furthermore, a sliced output variable can be used in only one nested for-loop. In the example, the code on the left does not work because j and k loop over different values; the code on the right works to index different portions of the sliced array A.

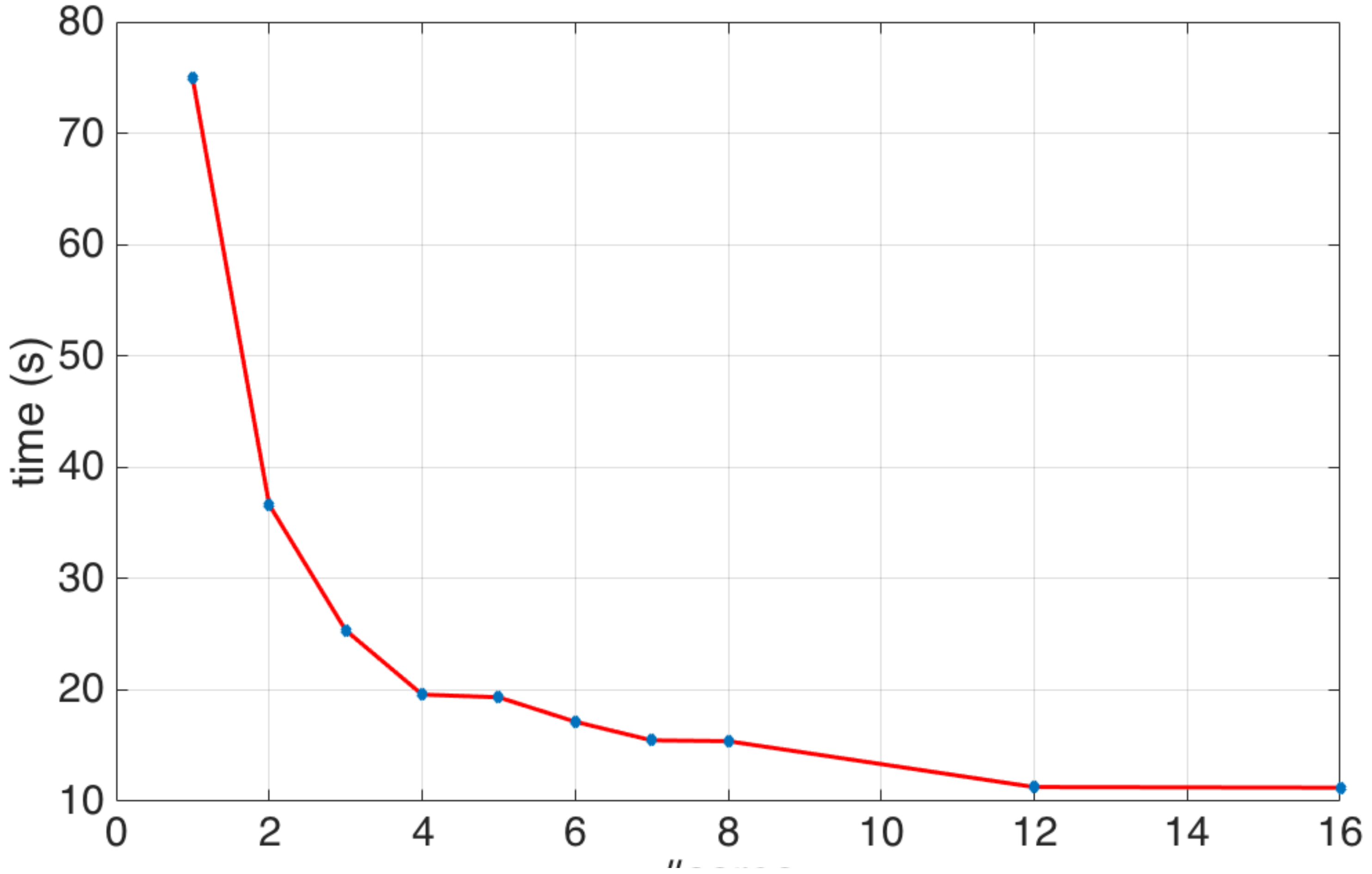
Example: parameter sweep

- Offload parameter sweep to local workers
- Get peak value results when processing is complete
- Plot results in local MATLAB





Parameter sweep speedup



feval - parfeval

feval – evaluate function

```
fun = 'round';
x1 = pi;
y = feval(fun,x1)
>>y=3
x2 = 2;
y = feval(fun,x1,x2)
>>y=3.1400
```

parfeval - Execute function asynchronously on parallel pool worker

```
f = parfeval(p,@magic,1,10);
value = fetchOutputs(f);
```

Parfeval example

```
n = 10000000;
job = cell(1,6);
for idx = 1:6
   jobs(idx) = parfeval(pool, @test, 1, n, idx);
end

% wait for outputs as they finish
output = cell(1, 6);
for idx = 1:6
   [completedIdx, value] = fetchNext(jobs);
   output{completedIdx} = value;
end
delete(pool);
```