Topics:
1. Parallel for-loops
   Loop-level, shared data
2. SPMD parallelization
   Same task, multiple data
3. Task parallelization
   Different task, multiple data

Parallel for
% Request 4 workers (max 12)
>> matlabpool 4
% Run in parallel
parfor i=1:n
   A(i)=func(B(i));
end
% Return workers
>> matlabpool close

Classification of variables

<table>
<thead>
<tr>
<th>Classification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop</td>
<td>Loop index for arrays</td>
</tr>
<tr>
<td>Sliced input</td>
<td>Array whose elements are read in parallel by different workers</td>
</tr>
<tr>
<td>Sliced output</td>
<td>Array whose elements are written in parallel by different workers</td>
</tr>
<tr>
<td>Broadcast</td>
<td>A variable defined before parallel and used inside parallel, but never assigned</td>
</tr>
<tr>
<td>Reduction</td>
<td>A variable that is accumulated in parallel</td>
</tr>
<tr>
<td>Temporary</td>
<td>A variable that is created inside parallel, not available outside parallel</td>
</tr>
</tbody>
</table>

Variables continued

twopi = 2*pi;
sum=0;
A=rand(100,1);
parfor i=1:100
   c=1.0/i;
   sum=sum+c;
   B(i)=c*twopi*A(i);
end

Note: A worker has its own memory space, all variables must be "communicated" to/from the client. A sliced variable is only communicated on the part that is used by the worker.

Communication is handled automatically (hidden from user) and can destroy the performance if you are not careful on how you access the data.

Workers cannot communicate peer to peer, i.e., update of B(i+1)=... is not allowed. Workers communicate only with client.
Parfor example MxM

```matlab
parfor i=1:n
    for j=1:n
        for k=1:n
            C(i,j)=C(i,j)+A(i,k)*B(k,j);
        end
    end
end
```

Note: C matrix is both sliced input and output. Lot of communication to/from client-worker (both ways)!

Parallel Performance

**Speedup:** $S = T_1 / T_P$ where $T_1$ is the run-time using one worker and $T_P$ using $P$ workers.

Parallel vs Serial

Note: Now $C$ is only sliced output, $d$ is a temporary.

Improved version MxM

```matlab
parfor i=1:n
    for j=1:n
        d=0;
        for k=1:n
            d=d+A(i,k)*B(k,j);
        end
        C(i,j)=d;
    end
end
```

Small overhead in reading $A$ and $B$, writing $C$

Parallel Performance

Single Program Multiple Data (SPMD)

```matlab
% Request 4 workers (max 12)
>> matlabpool 4
% Run in parallel on 4 workers
spmd (4)
    < statements >
end
% Return workers
>> matlabpool close
```
Variables in SPMD

- Each worker can read data from client defined outside spmd (replicated data).
- All data assigned inside spmd are composite (private) but can be accessed from client.
- Large data sets can be distributed over workers (distributed array).

Composite (private) data

```
A=rand(100,1);
spmd (N)
if (labindex==1)
    B=zeros(100,1);
else
    B=pi*A;
end
end
B1=B{1}; % Worker 1's data
B2=B{2}; % Worker 2's data etc.
```

Distributed data

```
A=rand(1,100);
Adist=distributed(A);
spmd
    for i=drange(1:100)
        B(i)=pi*Adist(i);
    end
end
B=gather(distributed(B));
```

The distributed arrays are split into different partitions (private data) and assigned to the different workers. The function `drange` picks out each partitions iteration indexes.

Performance in SPMD

Use composite data (private) and replicated arrays. Reading and writing distributed arrays takes very long time!
Bug in installation or bad design!??
Also, workers can not communicate peer to peer! (Grrhh...)
```
for i=drange(1:100)
    B(i)=pi*Adist(i+1);
end
```
Not allowed!!!

=> Can not have data dependencies between different partitions.

Task parallelism

Can create independent tasks (defined as Matlab functions) and schedule them to available workers (cores).
Can define arbitrary number of tasks (not limited to 12) and let the system schedule and load balance the work.
Note, we use functions for tasks. Then all data are local and private in the workers.
% create scheduler
sched=FindResource('scheduler','type','local');
joblist=createJob(sched); % Create a job queue

% Insert tasks to the queue
task1=createTask(joblist,@matmul,1,{A B});
task2=createTask(joblist,@matmul,1,{A2 B2});
task3=createTask(joblist,@matinv,1,{A});
submit(joblist); % submit the job

% wait for task2
waitforstate(task2);
res=get(task2,'OutputArguments');
c2=Res{1};

% wait for all tasks
waitforstate(joblist);
results=getAllOutputArguments(joblist);
c1=results{1};
c3=results{3};
destroy(joblist); % destroy the job queue

---

**Summary**

- Three constructs for parallelism
  - For-loops with parfor
  - Single Program Multiple Data
  - Task parallelism with scheduling
- Private memory on the workers
  (Can distribute and replicate data but not access other workers data)
- Matlab good for parallel independent tasks
  (e.g. parameter sweeps).

---

**Parallel Matlab at IT-dep**

Matlab Parallel Computing Toolbox is available on Gullviva.

Can only be run from Sunray stations
(computer rooms at 5.th floor).

Log in to a Sunray and open terminal, type
> xlogin gullviva

Start Matlab in the new terminal
> matlab