NTNU – Trondheim
Norwegian University of
Science and Technology

Introduction to Parallel Matlab (MDCS/Parallel Computing Toolbox)

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

Plan for the day

- Matlab multithreading functions and operators
- How to write faster code; basic examples
- Matlab vs Python operators
- Parallel Computing Toolbox and MDCS
- Introduction to parallel programming
- Tutorial 1. Parallel region (Intro, Hello world).
- Tutorial 2. Parallel loop
- Tutorial 3. Parallel region (Distributed arrays, Message passing).
- Tutorial 4. Implement C code to Matlab.
- Distributed Matlab using MPI
- Parallel R and Python
- Example with Deep Learing

All Matlab examples are done with Matlab R2016b, gcc/4.9.3 and cluster Maur (Intel E5-2670 2.6GHz: 32 GB, 2 x 8cores processors each node)

Matlab support multithreading for number of functions and operators.

(see eg. http://www.mathworks.com/support/solutions/en/data/1-4PG4AN/?solution=1-4PG4AN)

<u>List of some functions/operators:</u>

Functions that speed up for double arrays > 20k elements

Trigonometric: ACOS(x), ACOSH(x), ASIN(x), ASINH(x), ATAN(x), ATAND(x), ATANH(x), COS(x), COSH(x), SIN(x), SINH(x), TAN(x), TANH(x)

Exponential: EXP(x), POW2(x), SQRT(x)

Operators: X*Y (Matrix Multiply), X^N (Matrix Power)

Reduction Operations : MAX and MIN (Three Input), PROD, SUM

Matrix Analysis: DET(X), RCOND(X), HESS(X), EXPM(X)

Linear Equations: INV(X), LSCOV(X,x), LINSOLVE(X,Y), A\b

How to write faster code

(See more here:http://www.ee.columbia.edu/~marios/matlab/Writing_Fast_MATLAB_Code.pdf)

1. Preallocation:

You will see better performance to preallocate all arrays before using them as:

A) Not allocated array

```
N=100000000;
t=0;
for i=1:N
   A(i)=sin(t);
   t=t+0.01;
end
```

B) Preallocated array:

```
N=100000000;
t=0;
A=zeros(N,1);
for i=1:N
   A(i)=sin(t);
   t=t+0.01;
end
```

Run time on Maur: A) 10.3 sec B) 4.4 sec

How to write faster code ...

2. Array iteration:

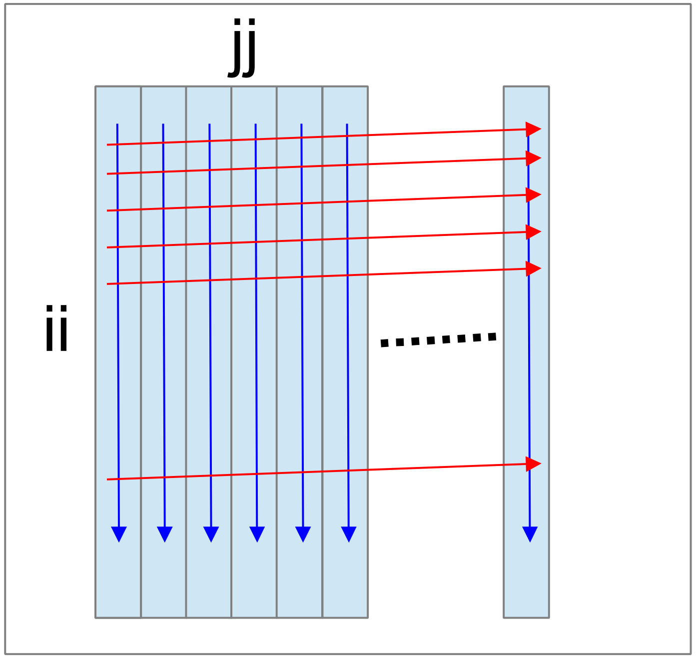
A) Row-wise iteration

```
A=rand(N,N);
avg1=0;
for ii=1:N
   for jj=1:N
      avg1=avg1+A(ii,jj)
   end
end
```

B) Column-wise iteration

```
A=rand(N,N);
avg1=0;
for jj=1:N
   for ii=1:N
      avg1=avg1+A(ii,jj)
   end
end
```

In Matlab: Arrays are organized cloumn-wise



Run time on my laptop: N=20000 A) 17 sec B) 10 sec

How to write faster code code ...
3. Vectorization:

A) Standard code

```
t=0;
N=100000000;
A=zeros(N,1);
for i=1:N
      A(i)=sin(t);
      t=t+0.01;
end
```

B) Vectorization:

```
t=0:0.01:N/100;
A=sin(t);
```

Running time on Maur: A) 4.4 sec B) 0.7 sec

Note! You can also write e.g. A=sin(t)+cos(t);

How to write faster code code ...
4. meshgrid:

Meshgrid create a grid of nodes with X and Y values.

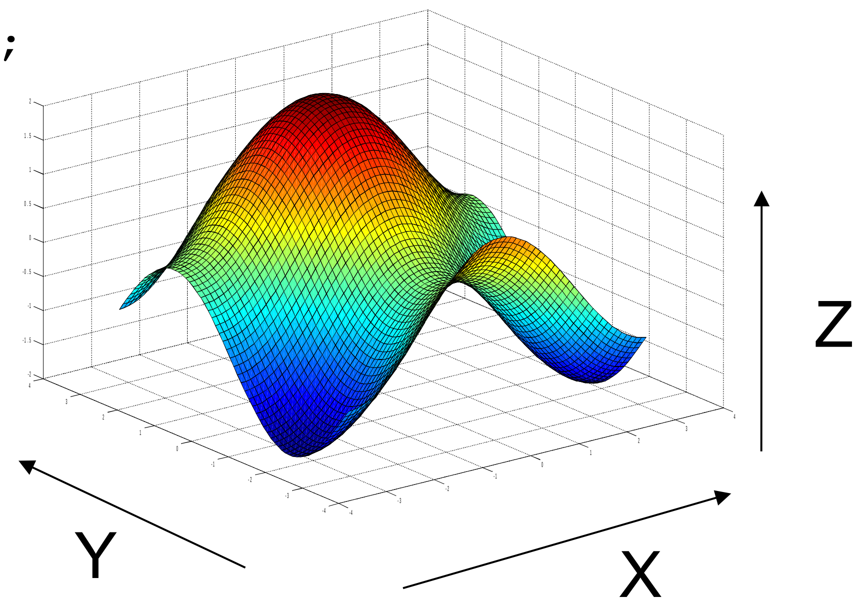
```
xstep=0.1; ystep=0.1;
Xmax=pi(); Xmin=-pi();
Ymax=pi(); Ymin=-pi();

[X,Y]=meshgrid(Xmin:xstep:Xmax,Ymin:ystep:Ymax);

% Creat Z value for each X and Y values.
Z=cos(X)+sin(Y);
```

Print out

```
surf(X,Y,Z);
```



How to write faster code code ...
4. meshgrid ...
Same code with for-loops:

```
%Standard for loop
Xstep=0.1; ystep=0.1;
Xmax=pi(); Xmin=-pi();
Ymax=pi(); Ymin=-pi();
xsteps=ceil((Xmax-Xmin)/xstep+xstep);
ysteps=ceil((Ymax-Ymin)/ystep+ystep);
xvalue=Xmin; yvalue=Ymin;
X=zeros(xsteps,ysteps);
Y=zeros(xsteps,ysteps);
Z=zeros(xsteps,ysteps);
for y=1:ysteps
    for x=1:xsteps
        X(x,y)=xvalue;
    end
    xvalue=xvalue+xstep;
end
for x=1:ysteps
    for y=1:xsteps
        Y(x,y)=yvalue;
    end
    yvalue=yvalue+ystep;
end
xvalue=Xmin; yvalue=Ymin;

for y=1:ysteps
    for x=1:xsteps
        Z(x,y)=cos(xvalue)+sin(yvalue);
        xvalue=xvalue+xstep;
    end
    yvalue=yvalue+ystep;
end
```

How to write faster code ...
4. meshgrid ....

Performance:
Input:

```
xstep=0.0005;
ystep=0.0005;
Xmax=pi();
Xmin=-pi();
Ymax=pi();
Ymin=-pi();
```

(Note! Do not use the surf(X,Y,Z) function with this inputs on your computer. It will take all the memory)

Run time (on Maur):

For-loop: 14.9 sec

Meshgrid: 0.8 sec

Matlab vs Python operators

Numpy and scipy are python library for array and matrix manipulation.

Numpy and scipy library increase the performance.

Both Matlab and numpy/scipy uses LAPACK librarys.

There are no license cost for numpy and scipy.

See:
https://www.hpc.ntnu.no/display/hpc/Python+Numpy+Scipy+and+Odespy

See
Lapack: http://www.netlib.org/lapack/

	Matrix multiplication C=A*B (nxn)		Inv matrix B=A\C		FFT
n	5000	10000	5000	10000	10000
Matlab operator	2.4s	15.9s	2.2s	11.5s	0.25s
Matlab for-loop	21 min	2h 38 min			
Scipy operator	13.3 s	1min 45s	28.2 s	218.4s	3.0s
Python for-loop	30h 53min	>10 days			
C for-loop (-O3)	1 min 38 s	16 min 43s			

Matlab ver: R2016b (Maur),
Vilje: Python/Scipy (intel mkl): v.2.7.9, intel.comp v.15.0.1, mpt v.2.10

Parallel Matlab

PCT(Parallel Computing Toolbox) is a separate part of a Matlab Client features and was available from version R2008a.

Distributed Matlab using MPI on Vilje (NTNU solution).

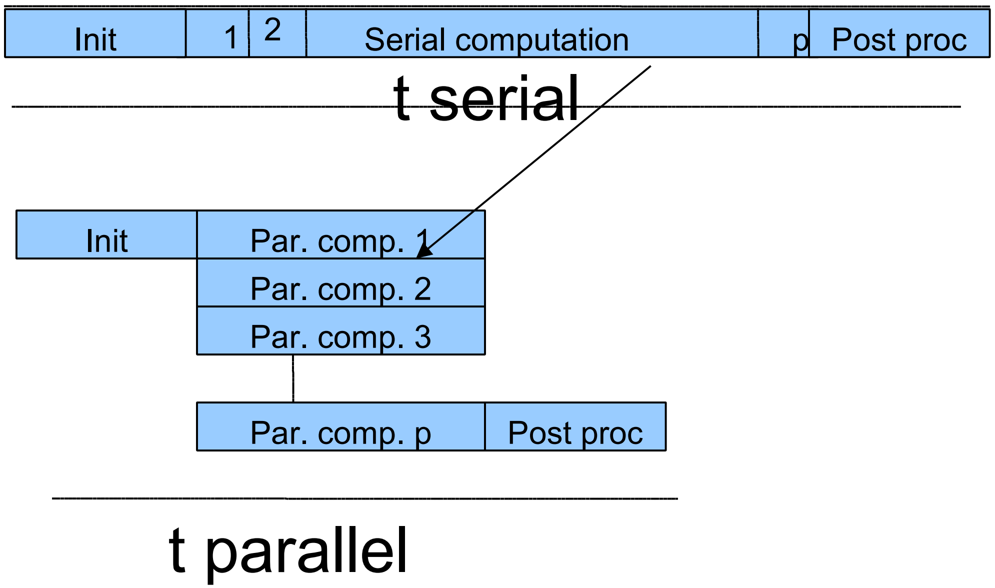
Programming Matlab using MPI

https://www.hpc.ntnu.no/pages/viewpage.action?pageId=15794234

Parallel computation

A program can be split up to run on several processors that runs in parallel.

Sequential program:



Speedup

S = t serial / t parallel

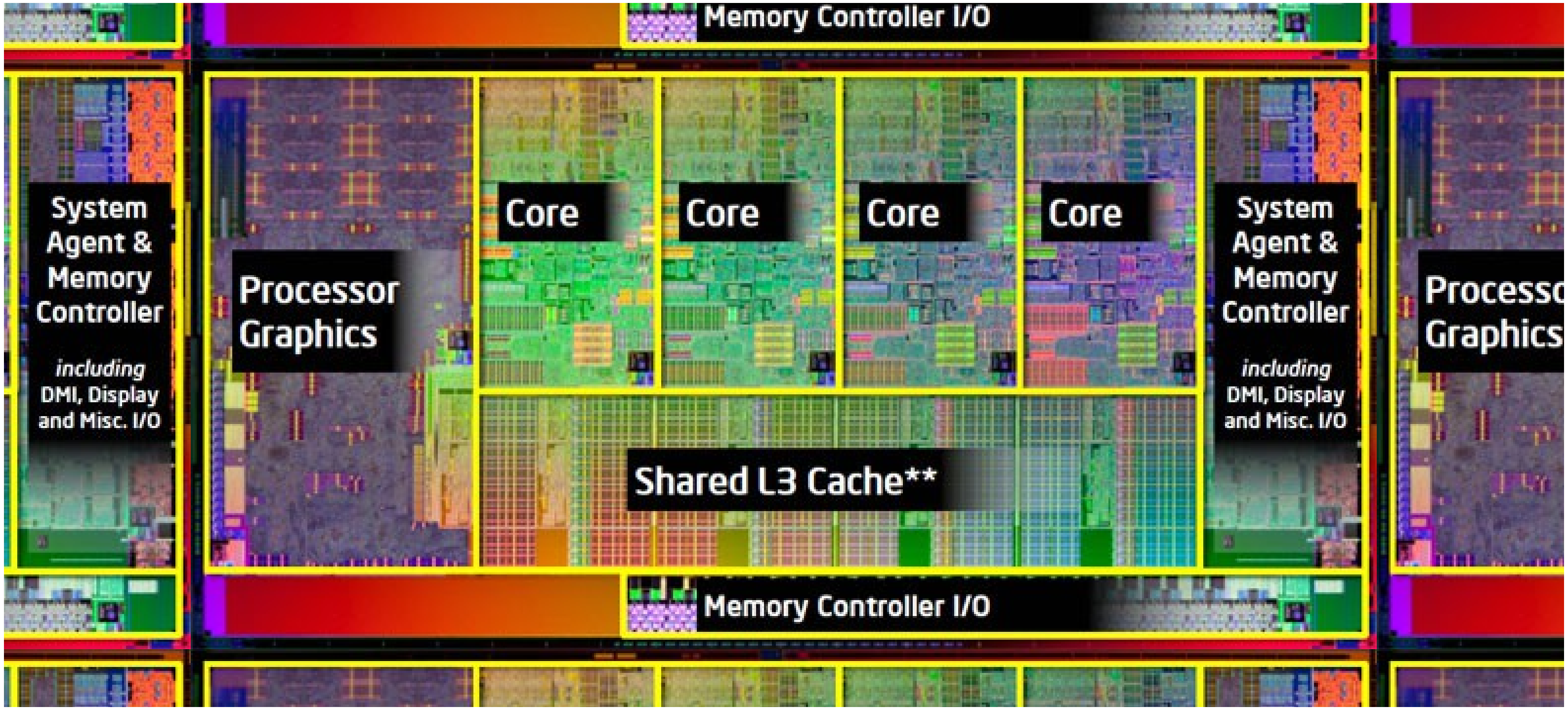
(t-serial: Execution time for a single core/processor progam

t-parallel:Execution time for the multicore/multiprocessor program)

Speedup for p processors/cores:

$S \leq p$.

Multicore shared memory processor.



Quad-core processor.

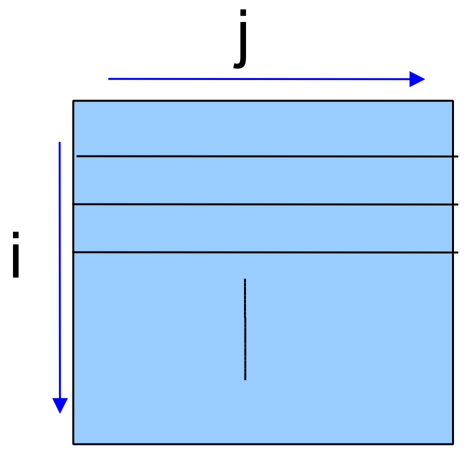
- Each core can run there own program block (thread), and simultaneously with the other cores.
- All cores share all the memory, and with fast memory access.
- All communication between the threads are via variables (shard memory).

Example: Matrix calculation.

B = c * A, where A and B is mxn matrices and c is a constant

Sequential computation:

All computation is carry out on only one processor or core.



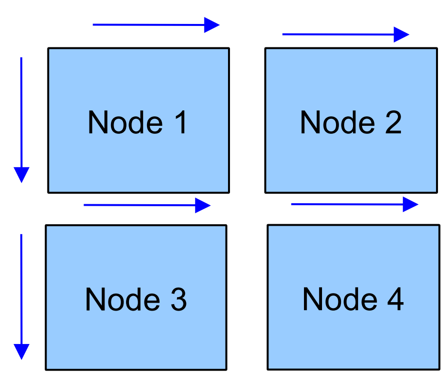
Program
Init the matrix A
for i = 1 to m
    for j = 1 to n
        B(i,j) = c * A(i,j)

Benefits: OK for small computation, fast memory access and none conflicts.

Drawback: Limited memory space (GB) and sequential computations.

Parallel computation with Message Passing.

The matrix is split up and scattered to several computers/nodes which are interconnected to each other via IP, infinity band or other high performance serial link.



Program:

Master: Initialize the matrix.
Master split up and spread the matrix to all nodes.

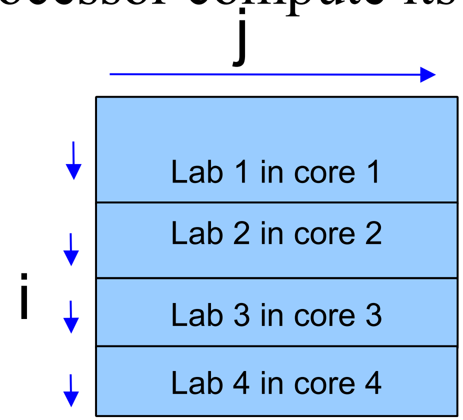
```
For i = 1 to m_mynode
    for j = 1 to n_mynode
        myB(i,j) = f(t) myA(i,j)
```

Benefits: More memory space (TB) and parallel computation on each node.

Drawback: Communication latency between the nodes.

<u>Parallel computation and shared memory.</u>

The matrix remains in the memory and each core/thread in the processor compute its part of the matrix in parallel.




```
Program
  parfor i = 1 to m
     B(i) = f(t) * A(i)
```

Benefits: Parallel computation and low communication latency between the cores.
Drawback: Small memory space (GB) and memory conflicts.

Matlab use a combination of this two last methods (Hybrid)

Tutorial 1. Parallel region. Lab creation.

A parallel region is the part of the program where program is spread in to several labs, cores and nodes. Before and after a parallel region the program run on 1 lab (master lab). It is called fork when the program go from 1 lab to parallel region and join when the program go back to 1 lab.
Matlab: worker and lab is the same

Get lab information:
labindex:
- Get the lab index (ID); which lab call the labindex .

numlabs:
- Number of labs/threads

Example

```
......
% Create a parallel pool.
parpool ( 'local' , 2) % local configuration on your computer and 2 labs

% Old setting: matlabpool open local 2

% 1 thread (Master thread: labindex=1)
......
spmd     % Fork to several labs (or cores) in parallel

     do_something_in_parallel();

end % Join to 1 thread

%Close parallel pool
delete(gcp('nocreate'));

%Old: matlabpool close
....
```

Synchronization: Barrier.

Each thread/labs waits until all threads/labs arrive.

Example Barrier

```
spmd    %parallel region

    do_many_things_in_parallel();

    //All threads wait here until all arrives.
    labBarrier;

    //All labs exchange its boundaries
    exchange_boundaries();

    //All threads wait here until all arrives.
    labBarrier;
    do_many_other_things_in_parallel();
end;
```

Exercise 1. Hello world.

Modify the sequential “Hello world” program in the file helloworld.m, and print out number of labs and lab number like this:.

“Hello world from lab 1 of 4”

How to display text and number:

Ex.

```
x=1;y=2;
disp ( ['Text ' , num2str(x) , ' text ' , num2str(y)] );
```

Out: Text 1 text 2

Tutorial 2. Parallel for-loop and data sharing.

Matlab automatically split up the for loop to several threads and send a copy of the block to each core with parfor. This construction is called worksharing, and shall be initialize as this:

```
parfor i=1:n
  % do_someting_in_parallel
end
```

Example parfor: 4 labs and n=40

Matlab divide the for loop into chunks, and the chunk size is 10.

```
parfor ii=1:n
    ....
end
```

lab 1	lab 2	lab 3	lab 4
for ii=1 to 10	for ii=11 to 20	for ii=21 to 30	for ii=31 to 40
...	...	...	...

Note! It is important that the parallel for loop is iterational independent. (This is not allowed A(i)=A(i-1));

That means; one iteration is independent of the iteration before. Parallel loop iterations are not in sequential order.

(Example fibonacci.m)

Data sharing: Shared and Private variables

In Matlab you do not see which variables are shared and private, but Matlab gives you a warning.

Shared

All variables declared outside a parallel region is shared inside the parallel region and private when written to.

Private

Variables declared inside the parallel region is private.

Note! The **parfor** iterator (eg "i") is set to private inside the parallel region.

Example 1. Private.

```
 .....
x=100; %x is shared variables between all cores/threads/labs
parfor i=1:n
    tmp =A(i);     % tmp is a private variable inside the parfor
    if tmp > 100
        B(i) = tmp - x;
    else
        B(i) = tmp;
    end
 end
 .....
```

You can not get the private variabel tmp outside the parallel region.

Tutorial 3. Reduction.

The matlab can reduce a shared variable inside a parallel for loop with an operator.

Example Average

```
n=100;
%Put random numbers into the vector v
v=rand(1,n);
ave=0;
parfor i=1 : n
    ave = ave + v(i);
end ;
ave = ave / n;
```

See ex_average.m

Other reduction operators:

+

-

*   /   .*

&

|

(See http://www.mathworks.com/help/toolbox/distcomp/brdqtjj-1.html#bq_of7_-3)

Exercise 2.

Calculation of Π (3.14159265358979...).
To calculate pi we can use this formula

$$\int_{0}^{1} 4/(1+x^2)\, dx = \Pi$$

Create a parallel version of the pi.m

Calculate the speedup S (Measure execution time before and after parfor loop (tic and toc)).

Use of arrays (perfomance)

Exampel: Average of an 2D array:

1. Standard for loop

```
A=rand(n,n);
avg1=0;
for ii=1:n %or parfor
   for jj=1:n
      avg1=avg1+A(ii,jj);
   end
end
```

n	10000	20000	30000
Sequential	8 sec	44	89
Parallel	5	21	53

2. With tmpA array

```
avg1=0;
for ii=1:n %or parfor
   tmpA=A(ii,:);
   for jj=1:n
      avg1=avg1+tmpA(jj);
   end
end
```

n	10000	20000	30000
Sequential	6 sec	29	64
Parallel	1.7	11	27

Use of arrays

Exampel: Average of an 2D array:

Sequential:

<u>3. Switched index ii and jj</u>

```
A=rand(n,n);
avg1=0;
for jj=1:n %or parfor
   for ii=1:n
      avg1=avg1+A(ii,jj);
   end
End
```

n	10000	20000	30000
Sequential	2.7	11	24
Parallel	3.8	21	50

<u>4. One dim array</u>

```
A=rand(1,n*n);
avg1=0;
for ii=1:n*n %or parfor
      avg1=avg1+A(ii);
end
```

n	10000	20000	30000
Sequential	2.6	10	24
Parallel	2.3	13	32

Tutorial 4. Distributed Arrays

Matlab distribute an array between all labs, also between nodes on a cluster.

Example:

Array size = 100, and number of labs are 4, the partition is 24 elements each lab.

```
A=ones(N,N);
A=distributed(A);  %Set outside the parallel region
spmd %Parallel region
    A=A*labindex
end
```

Codistributed array shall be set inside parallel region.

Composite distribute objects to all labs

(See example ex_distr_array.m)

Message Passing

Matlab have several message passing functions as:

labSend

labReceive

labSendReceive

labBroadcast.

(**See** http://www.mathworks.com/help/toolbox/distcomp/f1-6010.html)

Example labSendReceive (nonblocking function)

Syntax
data_received = labSendReceive(labTo, labFrom, data_sent)
data_received = labSendReceive(labTo, labFrom, data_sent, tag)

Arguments
data_sent
    Data on the sending lab that is sent to the receiving lab; any MATLAB data type.
data_received
    Data accepted on the receiving lab.
labTo
    labindex of the lab to which data is sent.
labFrom
    labindex of the lab from which data is received.
tag
    Nonnegative integer to identify data.

See ex_sendrec and ex_sendmatr

Implement C code to your Matlab program

Benefits for implement c code to your matlab code is faster code.

How to do this:

1. Create a c file, for your c function, as myfunction.c.
2. Include “mex.h” in top of your program.
3. The c file must contain:
   Your function and the mexFunction.
4. Compile your code with: mex myfunction.c
   Matlab create a mexa64 file as myfunction.mexa64.
5. Add your c code to your matlab code as
   >>out = myfunction (in1,in2,...,inN)

Example ex_mexfile

Mex function -interface between matlab and c-code

```
void mexFunction( int nlhs, mxArray *plhs[],
         int nrhs, const mxArray *prhs[] )
```

nrhs:	Number of input parameters
nlhs:	Number of output parameters
*prhs[]:	pointer to input parameters
*plhs[]	pointer to output parameters

Get a parameter

```
double x  = (double) mxGetScalar(prhs[0]);
```

Get a pointer (to an array)

```
double *v = (double *) mxGetPr(prhs[0]);
```

Create a matrix for the return argument

```
plhs[0] = mxCreateDoubleMatrix(1, 1, mxREAL);
```

```
//Sequential
static void mysincos( long n, long m,  double *y )
{
   double ly=0;
   long i,j;
   double Pi=3.141592653589793;

   for (i=1;i<=n;i++)
       for (j=1;j<=m;j++)
            ly=ly+cos((2*Pi*j)/m)*sin((2*Pi*j)/m);
   *y=ly;
   return;
}
//With OpenMP
static void mysincos( long n, long m,  double *y)
{
   double ly=0;
   long i,j;
   double Pi=3.141592653589793;
   #pragma omp parallel for private(i,j) reduction(+:ly)
   for (i=1;i<=n;i++)
       for (j=1;j<=m;j++)
            ly=ly+cos((2*Pi*j)/m)*sin((2*Pi*j)/m);
   *y=ly;
    return;
}
```

Compiling with Openmp code

mex CC=gcc CFLAGS="\$CFLAGS -fopenmp" LDFLAGS="\
$LDFLAGS -fopenmp" mysincos_omp.c

Vilje and Idun:
R2016b: module load gcc/4.9.x

Compare time consumption for matlab, C, and matlab with including mexfunction.
(see ex_mexfile.m)

Cores	Matlab (parfor)	Matlab mex c openmp	C openmp (O2)
(sequential) 1	1 min 23 sec	28 sec	28 sec
8	11 sec		3.54 sec
16	5.4 sec	2.1 sec	2.0 sec

Matlab R2016b , gcc v 4.9.1
n=m=30000;