



NTNU – Trondheim
Norwegian University of
Science and Technology

Introduction to Parallel Matlab (MDCS/Parallel Computing Toolbox)

NTNU-IT HPC Section
John Floan

Plan for the day

- Matlab multithreading functions and operators
- Matlab vs Python
- 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 .

Matlab support multithreading for number of functions and operators.

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

List of some functions/operators:

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

Matlab vs Python

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

Numpy and scipy library increase the performance.

Both Matlab and numpy/scipy use LAPACK librarys.

There are no license cost for numpy and scipy.

Scipy and numpy:

www.scipy.org

<http://docs.scipy.org/doc/scipy/reference/>

Lapack: <http://www.netlib.org/lapack/>

Performance test on Kongull .

	Matmult: $C = A * B$		Inv: $B = A \setminus C$		FFT
n	2000	5000	2000	5000	5000
Matlab Operator	0,2s	3,1s	0,7s	6,5s	0,6s
Matlab for-loop	499s (8 min)	9524s (2.6 h)			
Matlab Parfor	920s (15min)				
Scipy Operator	2,5s	38s	6,2s	93s	1,3s
Python For-loop	16202 (4.5h)	>24h			
C Openmp (12 cores)	66s	169s			

Note: Scipy, on kongull, runs on 1 core.

Matlab (MDCS/Parallel Computing Toolbox).

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

Latest available Matlab version on NTNU is R2010b.

Allows up to 8 labs and cores.

(Workers and labs are same as threads, and each thread is distributed on a single independent cpu core).

MDCS (Matlab Distributed Compute Server)

MDCS allows up to 256 workers and cores. (That is 21 nodes on Kongull).

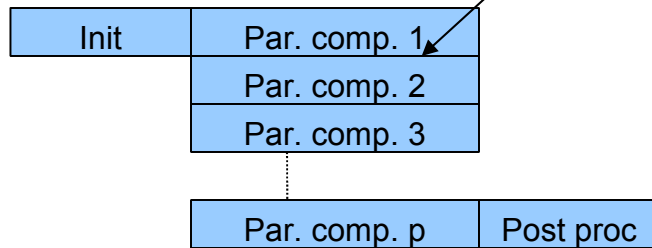
Parallel computation

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

Sequential program:



t serial



t parallel

Speedup

$$S = t_{\text{serial}} / t_{\text{parallel}}$$

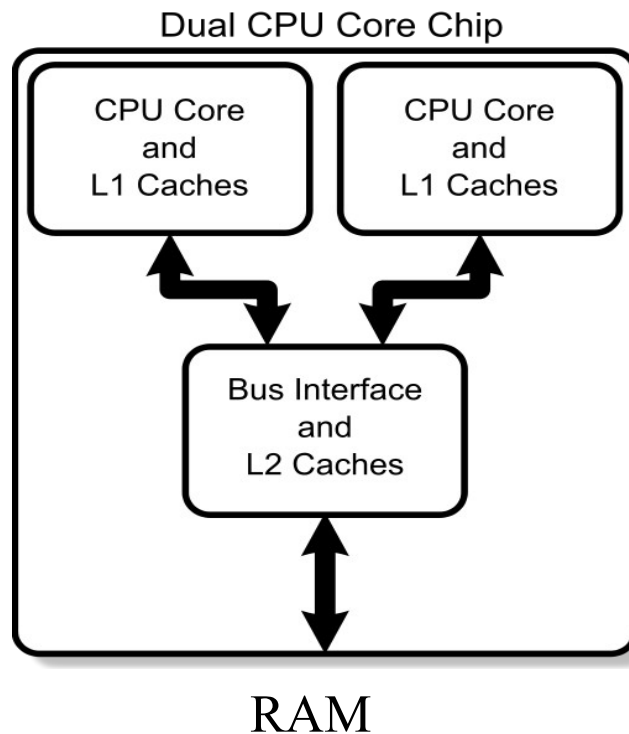
(t-serial: Execution time for a single core/processor program)

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

Speedup for p processors/cores:

$$S \leq p.$$

Multicore shared memory processor.



Dual-core processor with level 1 and 2 caches.

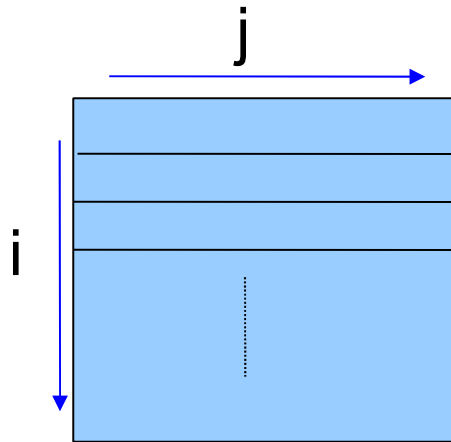
- Each core can run there own program block (thread), and simultaneously with the other cores.
- All cores share all the memory, and with fast memory access.
- All communication between the threads are via variables (shard memory).

Example: Matrix calculation.

$B = f(t) A$, where A and B is $m \times n$ matrices and $f(t)$ is a function

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

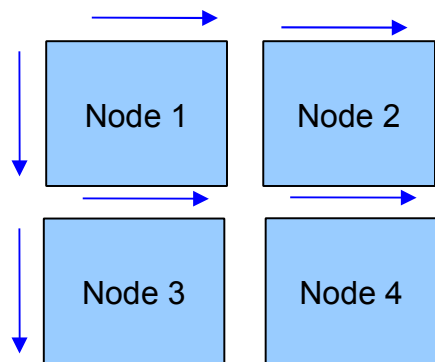
$B(i,j) = f(t) * A(i,j)$

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.

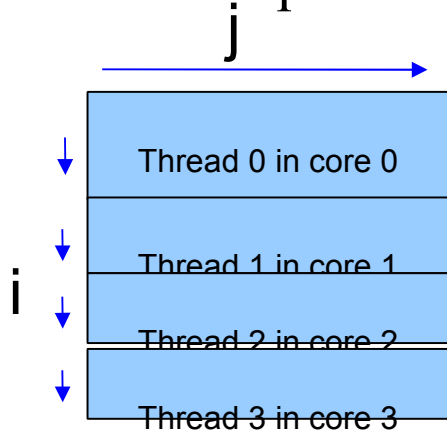
```
For i = 1 to m_mynode
  for j = 1 to n_mynode
    myB(i,j) = f(t) myA(i,j)
```

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

Drawback: Communication latency between the nodes.

Parallel computation and shared memory.

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)

Kongull

	Each Node	Total
Cores	12	1116
Nodes	-	93
Memory	48GB(44 nodes) 24GB(49 nodes)	3TB
Storage	-	73TB

Tutorial 1. Parallel region. Thread creation.

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

Matlab; a **thread** is called a worker or a lab

Get lab information:

labindex:

- Get the lab index (ID); which lab/thread call the labindex .

numlabs:

- Number of labs/threads

Example

.....

% 1 thread (Master thread: labindex=1)

.....

spmd % Fork to several threads/labs in parallel

do_something_in_parallel();

end % Join to 1 thread

....

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;
```

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 i=1:n
    ....
end
```

lab 1	lab 2	lab 3	lab 4
for i=1 to 10	for i=11 to 20	for i=21 to 30	for i=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. Matlab gives you a warning if this is wrong.

Shared

All variables declared outside a parallel region is shared inside the parallel region.

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 parfor.

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/brdqttj-1.html#bq_of7_-3)

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”

Exercise 2. labBarrier.

Modify your program to print out as this:

Number of labs are 4 (Always first)

Hello world from lab 1 of 4

Exercise 3.

Calculation of Π (3.14159265358979...).

To calculate pi we can use this formula

$$\int_0^1 \frac{4}{1+x^2} dx = \Pi$$

Create a parallel version of the pi.m

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

Note! Matlab parfor do not like ≥ 2 dim array (Older than R2011a):

```
X = rand(n,n);
Y = zeros(n,n);
parfor i = 1 : n
    for j = 1 : n
        Y(i,j) = k * X ( i , j ); %Do not work
    end
end
```

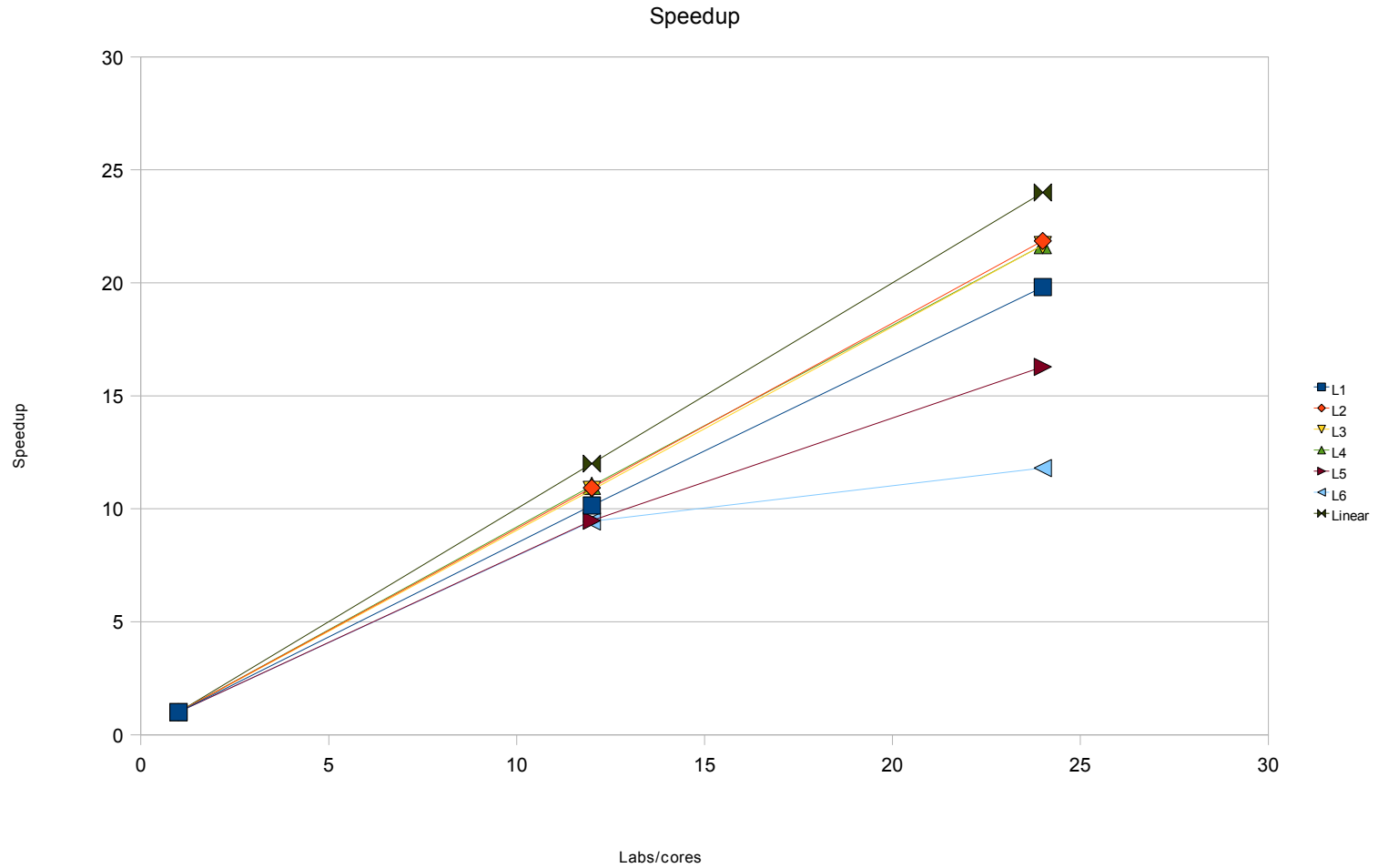
To go around this;

Split the array to a temporarily 1 dim array.

```
parfor i = 1 : n
    tX = X( i , : );
    tY = Y( i , : );
    for j = 1 : n
        tY(j) = k * tX ( j );
    end
    Y(i,:)=tY;
end
```

Performance on Kongull.

From ex_perf_parfor.m file. 6 Loops



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:

```
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);
```

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

Cores/labs	Matlab parfor (min)	Matlab mex c (min)	Matlab mex c par	Standard C openmp	Matlab / c
1	164	11	11	12	13
4	42	11	3	3	14
8	22	11	1	2	14
Speedup					
1	1,0	1,0	1,0	1,0	
4	3,9	1,0	4,0	4,0	
8	7,5	1,0	7,9	8,1	