Parallel MATLAB at VT: Parallel For Loops

Gene Cliff (AOE/ICAM - ecliff@vt.edu)
Justin Krometis (ARC/ICAM - jkrometis@vt.edu)
Wednesday, 21 October 2015
9:00am - 10:00am
..... NLI .....
AOE: Department of Aerospace and Ocean Engineering
ARC: Advanced Research Computing
ICAM: Interdisciplinary Center for Applied Mathematics

VirginiaTech

- **Introduction**
- FMINCON Example
- Executing a PARFOR Program
- PRIME Example
- Classification of variables
- ODE_SWEEP Example
- MD Example
- Conclusion

INTRO: Parallel MATLAB

In a previous lecture we discussed MATLAB's *Parallel Computing Toolbox* (**PCT**), and the *Distributed Computing Server* (**MDCS**) that runs on Virginia Tech's `Ithaca` cluster.

As noted previously there are three ways to write a parallel MATLAB program:

- suitable **for** loops can be made into **parfor** loops;
- the **spmd** statement can define cooperating synchronized processing;
- the **task** feature creates multiple independent programs.

Today we focus on **parfor** loops and on **options** for parallelism in MATLAB provided toolboxes.

VirginiaTech

INTRO: PARFOR: Parallel FOR Loops

Today's Lecture: PARFOR

The simplest path to parallelism is the **parfor** statement, which indicates that a given **for** loop can be executed in parallel.

When the "client" MATLAB reaches such a loop, the iterations of the loop are automatically divided up among the workers, and the results gathered back onto the client.

Using **parfor** requires that the iterations are completely independent; the results must not depend on the order of execution. There are also restrictions on array-data access.

OpenMP implements a `directive` for 'parallel for loops'

VirginiaTech

PROGRAMMING: "SPMD" Single Program Multiple Data

Lecture #3: SPMD Wednesday, 5 November, 3:30-4:40pm

MATLAB can also work in a simplified kind of MPI model.

There is always a special "client" process.

Each worker process has its own memory and separate ID.

There is a single program, but it is divided into client and worker sections; the latter marked by special **spmd/end** statements.

Workers can "see" the client's data; the client can access and change worker data.

The workers can also send messages to other workers.

OpenMP includes constructs similar to **spmd**.

VirginiaTech

- Introduction
- **FMINCON Example**
- Executing a PARFOR Program
- PRIME Example
- Classification of variables
- ODE_SWEEP Example
- MD Example
- Conclusion

FMINCON: Hidden Parallelism

FMINCON is a popular MATLAB function available in the Optimization Toolbox. It finds a local minimizer of a real-valued function of several real variables with constraints:

```
min F(X)  subject to:

A*X   <= B,
Aeq*X  = Beq  (linear constraints)
C(X)  <= 0,
Ceq(X) = 0    (nonlinear constraints)
LB <= X <= UB (bounds)
```

If no derivative (Jacobian) information is supplied by the user, then **FMINCON** uses finite differences to estimate these quantities. If **F, C** or **Ceq** are expensive to evaluate, the finite differencing can dominate the execution time.

VirginiaTech

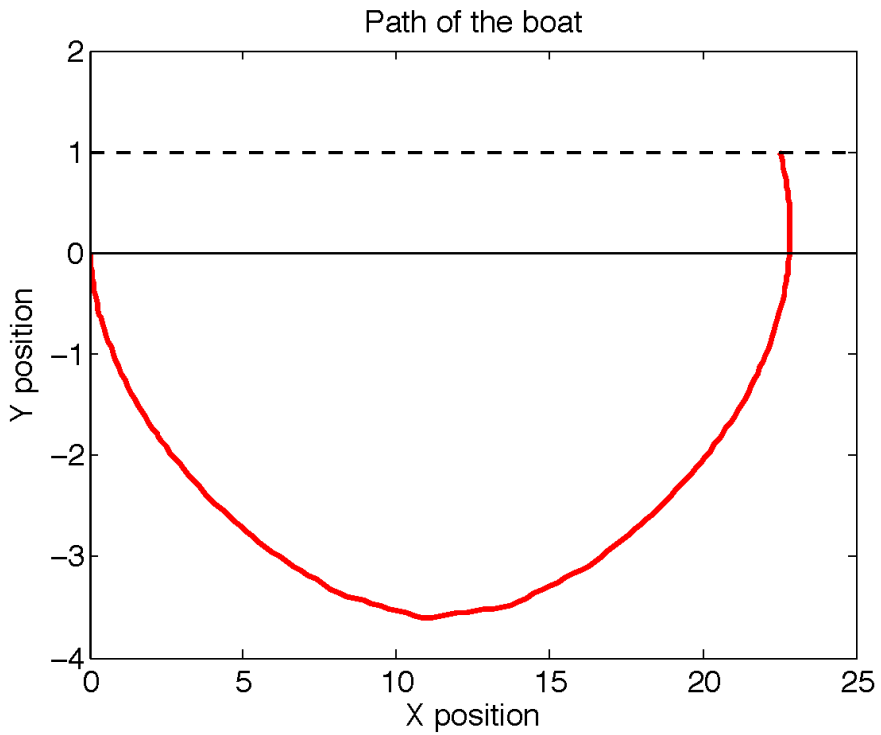
FMINCON: Path of a Boat Against a Current

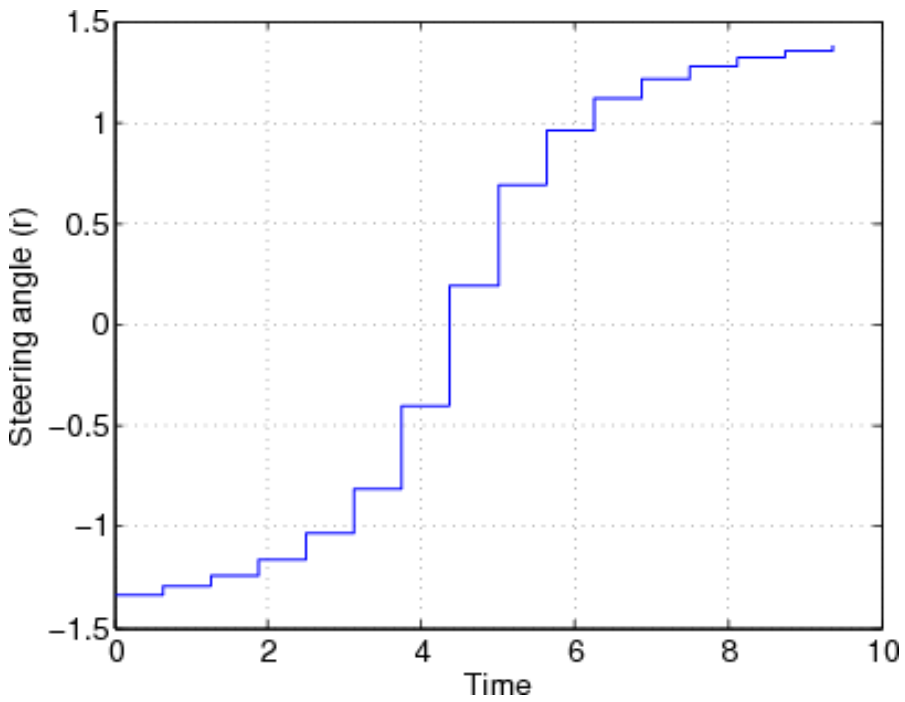
An example using FMINCON involves a boat trying to cross a river against a current. The boat is given 10 minutes to make the crossing, and must try to land as far as possible upstream. In this unusual river, the current is zero midstream, negative above the x axis, and positive (helpful) below the x axis!

VirginiaTech

FMINCON: Riding the Helpful Current

The correct solution takes maximum advantage of the favorable current, and then steers back hard to the land on the line $\mathbf{y = 1}$.





VirginiaTech

FMINCON: Hidden Parallelism

FMINCON uses an **options** structure that contains default settings. The user can modify these by calling the procedure **optimset**. The finite differencing process can be done in parallel if the user sets the appropriate option:

```
options = optimset ( optimset( 'fmincon' ), ...
                       'LargeScale','off', ...
                       'Algorithm', 'active-set', ...
                       'Display' , 'iter', ...
                     'UseParallel', 'Always');

[ x_star, f_star, exit ] = fmincon ( h_cost, z0, ...
  [], [], [], [], LB, UB, h_cnst, options );
```

In **version 2013a** and earlier, the user must invoke the **pool (matlabpool) command to make workers available!**

VirginiaTech

MATLAB Toolboxes that employ **PCT** in **ver 2013b**

- **Simulink**
- **Computational Biology**
- **Control System Design and Analysis**
- **Image Processing, Signal Processing**
- **Optimization**
- **Statistics**
- see http://www.mathworks.com/products/parallel-computing/builtin-parallel-support.html

VirginiaTech

- Introduction
- FMINCON Example
- **Executing a PARFOR Program**
- PRIME Example
- Classification of variables
- ODE_SWEEP Example
- MD Example
- Conclusion

INTRO: Direct Execution for PARFOR

Parallel MATLAB jobs can be run *directly*, that is, interactively.

The **parpool** command is used to reserve a given number of workers on the local (or perhaps remote) machine.

Once these workers are available, the user can type commands, run scripts, or evaluate functions, which contain **parfor** statements. The workers will cooperate in producing results.

Interactive parallel execution is great for desktop debugging of short jobs.

It's an inefficient way to work on a cluster, because no one else can use the workers until you release them!

Note: Starting in R2013b, if you try to execute a parallel program and a pool of workers is not already open, MATLAB will open it for you. The pool of workers will then remain open for a time that can be specified under Parallel → Parallel Preferences (default = 30 minutes).

INTRO: Indirect Execution for PARFOR

Parallel PARFOR MATLAB jobs can be run indirectly.

The **batch** command is used to specify a MATLAB code to be executed, to indicate any files that will be needed, and how many workers are requested.

The **batch** command starts the computation in the background. The user can work on other things, and collect the results when the job is completed.

The **batch** command works on the desktop, and can be set up to access ARC clusters (e.g. Ithaca).

VirginiaTech

INTRO: batch options

- 'Workspace' — A 1-by-1 struct to define the workspace on the worker just before the script is called. The field names of the struct define the names of the variables, and the field values are assigned to the workspace variables. By default this parameter has a field for every variable in the current workspace where batch is executed. This parameter supports only the running of scripts.
- 'Profile' — A single string that is the name of a cluster profile to use to identify the cluster. If this option is omitted, the default profile is used to identify the cluster and is applied to the job and task properties.
- 'AdditionalPaths' — A string or cell array of strings that defines paths to be added to the MATLAB® search path of the workers before the script or function executes. The default search path might not be the same on the workers as it is on the client; the path difference could be the result of different current working folders (pwd), platforms, or network file system access. The 'AdditionalPaths' property can assure that workers are looking in the correct locations for necessary code files, data files, model files, etc.
- 'AttachedFiles' — A string or cell array of strings. Each string in the list identifies either a file or a folder, which gets transferred to the worker.
- 'CurrentFolder' — A string indicating in what folder the script executes. There is no guarantee that this folder exists on the worker. The default value for this property is the cwd of MATLAB when the batch command is executed. If the string for this argument is '.', there is no change in folder before batch execution.
- 'CaptureDiary' — A logical flag to indicate that the toolbox should collect the diary from the function call. See the diary function for information about the collected data. The default is true.
- 'Pool' — An integer specifying the number of workers to make into a parallel pool for the job *in addition* to the worker running the batch job itself. The script or function uses this pool for execution of statements such as parfor and spmd that are inside the batch code. Because the pool requires N workers in addition to the worker running the batch, there must be at least N+1 workers available on the cluster. You do not have to have a parallel pool already running to execute batch; and the new pool that batch creates is not related to a pool you might already have open. (See Run a Batch Parallel Loop.) The default value is 0, which causes the script or function to run on only the single worker without a parallel pool.

VirginiaTech

- Introduction
- FMINCON Example
- Executing a PARFOR Program
- **PRIME Example**
- Classification of variables
- ODE_SWEEP Example
- MD Example
- Conclusion

PRIME: The Prime Number Example

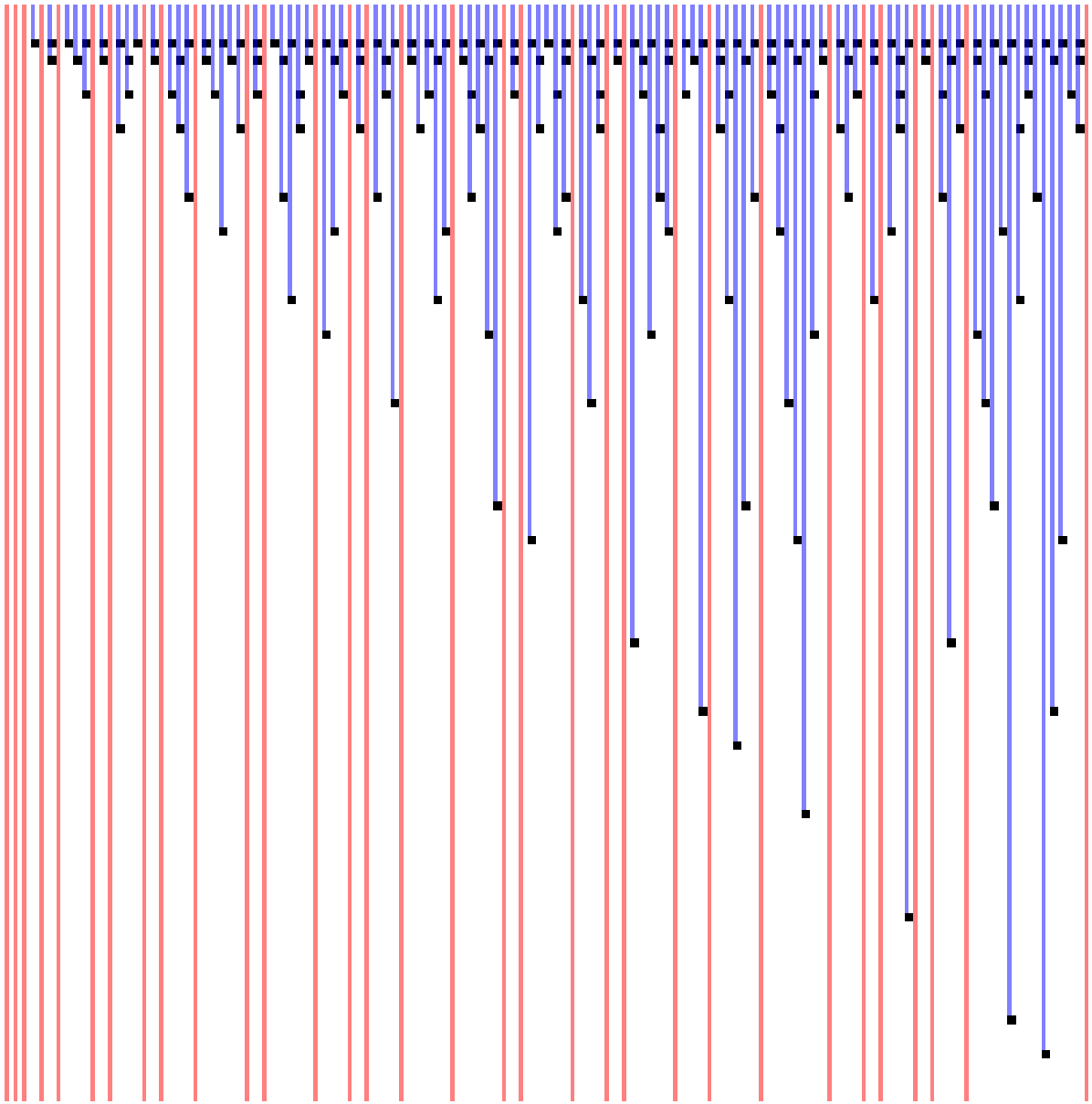
For our next example, we want a simple computation involving a loop which we can set up to run for a long time.

We'll choose a program that determines how many prime numbers there are between 1 and **N**.

If we want the program to run longer, we increase the variable **N**. Doubling **N** multiplies the run time roughly by 4.

VirginiaTech

PRIME: The Sieve of Eratosthenes



VirginiaTech

PRIME: Program Text

```
function total = prime ( n )

%% PRIME returns the number of primes between 1 and N.

  total = 0;

  for i = 2 : n

    prime = 1;

    for j = 2 : i - 1
      if ( mod ( i, j ) == 0 )
        prime = 0;
      end
    end

    total = total + prime;

  end

  return
end
```

VirginiaTech

PRIME: We can run this in parallel

We can parallelize the loop whose index is **i**, replacing **for** by **parfor**. The computations for different values of **i** are independent.

There is one variable that is not independent of the loops, namely **total**. This is simply computing a running sum (a **reduction variable**), and we only care about the final result. MATLAB is smart enough to be able to handle this summation in parallel.

To make the program parallel, we replace **for** by **parfor**. That's all!

Beware - **reduction assignments** require care

http://www.mathworks.com/help/coder/ug/reduction-assignments-in-parfor-loops.html?searchHighlight=parfor

VirginiaTech

PRIME: Local Execution With parpool

```
% matlabpool ( 'open', 'local', 4) % old  form
pool_obj = parpool('local', 4); % (R2013b)

n=50;

while ( n <= 5000000 )
    primes = prime_number_parfor ( n );
    fprintf ( 1, '__%8d__%8d\n', n, primes );
    n = n * 10;
end

%matlabpool  ( 'close' )
delete(pool_obj);
```

VirginiaTech

PRIME: Timing

```
PRIME_PARFOR_RUN
  Run PRIME_PARFOR with 0,  2, and 4 labs.

      N           1+0         1+2       1+4

   50000         0.13        0.10      0.08
  500000         3.25        1.66      0.85
 5000000         84.9        43.3      21.8
```

PRIME: Timing Comments

There are many thoughts that come to mind from these results!

This data suggests two conclusions:

Parallelism doesn't pay until your problem is big enough;

AND

Parallelism doesn't pay until you have a decent number of workers.

VirginiaTech

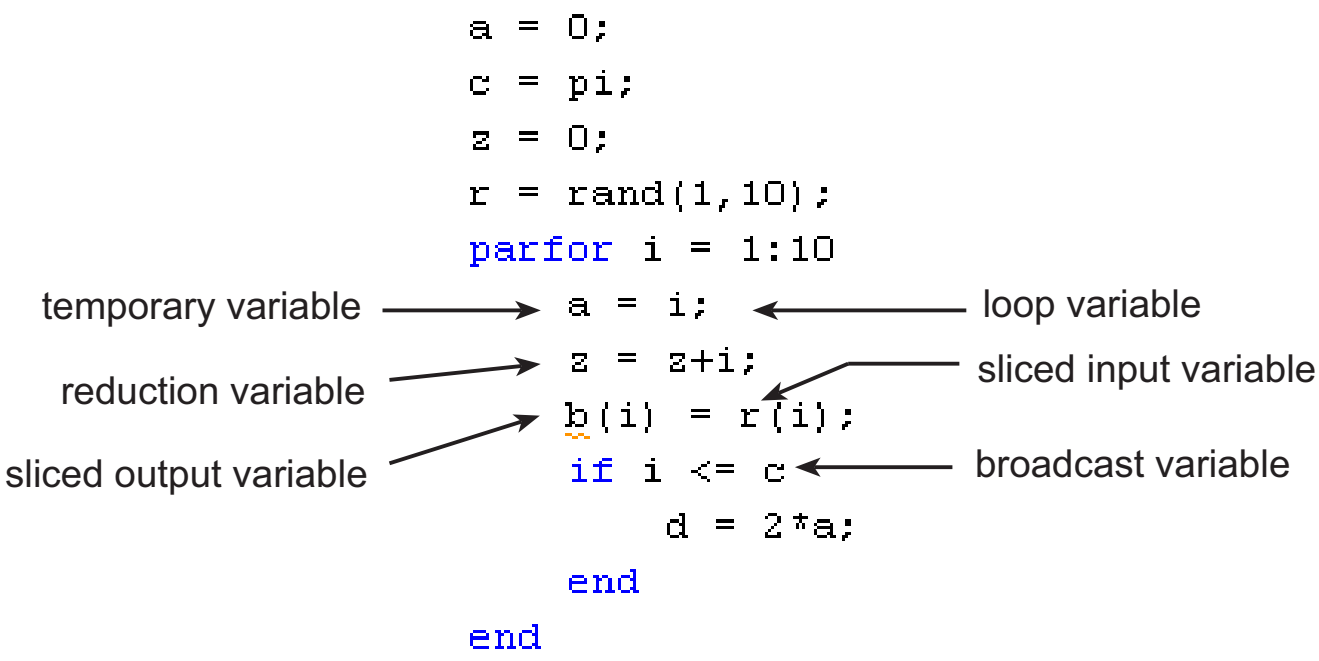
- Introduction
- FMINCON Example
- Executing a PARFOR Program
- PRIME Example
- **Classification of variables**
- ODE_SWEEP Example
- MD Example
- Conclusion

CLASSIFICATION: variable types in PARFOR loops

Classification	Description
Loop	Serves as a loop index for arrays
Sliced	An array whose segments are operated on by different iterations of the loop
Broadcast	A variable defined before the loop whose value is used inside the loop, but never assigned inside the loop
Reduction	Accumulates a value across iterations of the loop, regardless of iteration order
Temporary	Variable created inside the loop, but unlike sliced or reduction variables, not available outside the loop

VirginiaTech

CLASSIFICATION: an example



NB: "The range of a parfor statement must be increasing consecutive integers"

Trick Ques: What values to **a**, **i**, and **d** have after exiting the loop ?

VirginiaTech

Sliced variables:

Characteristics of a Sliced Variable. A variable in a `parfor`-loop is sliced if it has all of the following characteristics. A description of each characteristic follows the list:

- Type of First-Level Indexing — The first level of indexing is either parentheses, `()`, or braces, `{}`.
- Fixed Index Listing — Within the first-level parenthesis or braces, the list of indices is the same for all occurrences of a given variable.
- Form of Indexing — Within the list of indices for the variable, exactly one index involves the loop variable.
- Shape of Array — In assigning to a sliced variable, the right-hand side of the assignment is not `[]` or `''` (these operators indicate deletion of elements).

VirginiaTech

Allocating the loop indices

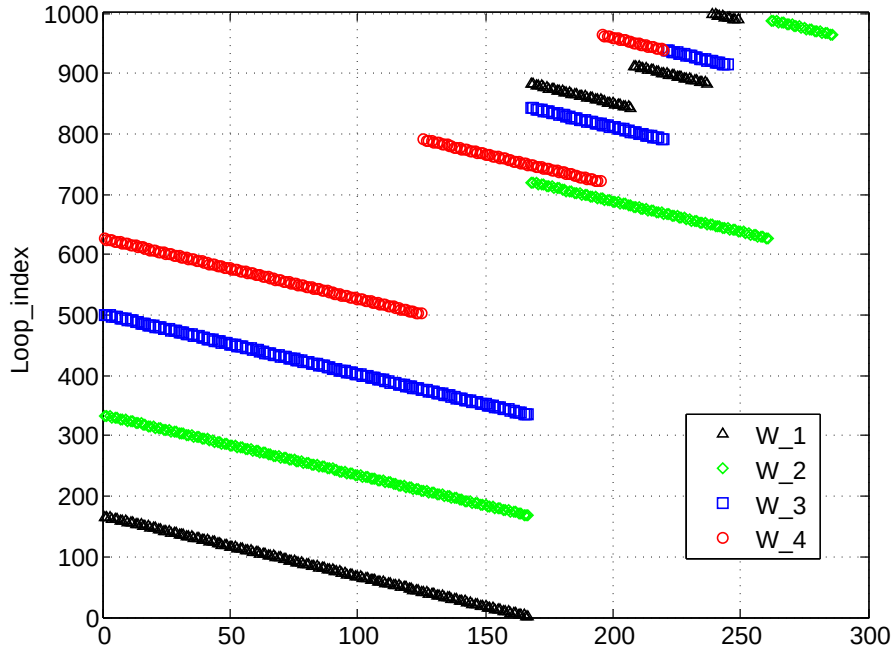
How are the *loop indices* distributed among the workers ?

Run *ii=1:1000* on 4 workers.

$\approx$ 63% indices allocated in the first 4 *chunks*.

Indices then assigned in smaller chunks as a worker finishes

Similar to **OpenMP**'s **Dynamic** assignment



VirginiaTech

PCT vs Multithread:

- At least since **R2007a** basic MATLAB has incorporated multithreading
- Many linear algebra functions (*e.g.* `LU`, `QR`, `SVD`) use multithreads
  (http://www.mathworks.com/support/solutions/en/data/1-4PG4AN/index.html?solution=1-4PG4AN)
- In the **PCT** each worker/lab runs a single thread
- For some codes using `parfor` will increase the execution time

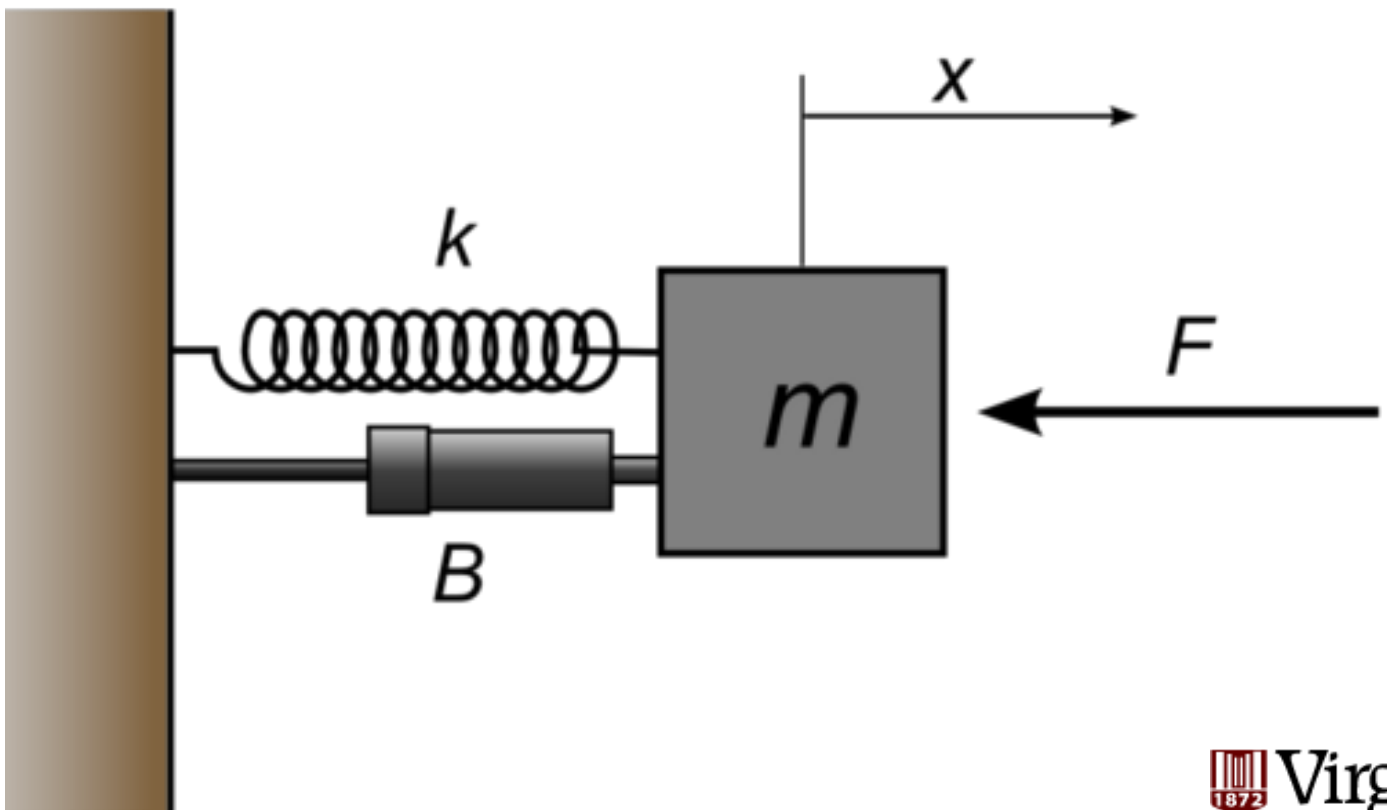
VirginiaTech

- Introduction
- FMINCON Example
- Executing a PARFOR Program
- PRIME Example
- Classification of variables
- **ODE_SWEEP Example**
- MD Example
- Conclusion

ODE: A Parameterized Problem

Consider a favorite ordinary differential equation, which describes the motion of a spring-mass system:

$$m\frac{d^2x}{dt^2} + b\frac{dx}{dt} + k\,x = f(t), \quad x(0) = 0,\ \dot{x}(0) = v\ .$$



VirginiaTech

ODE: A Parameterized Problem

Solutions of this equation describe oscillatory behavior; $x(t)$ swings back and forth, in a pattern determined by the parameters m, b, k, f and the initial conditions.

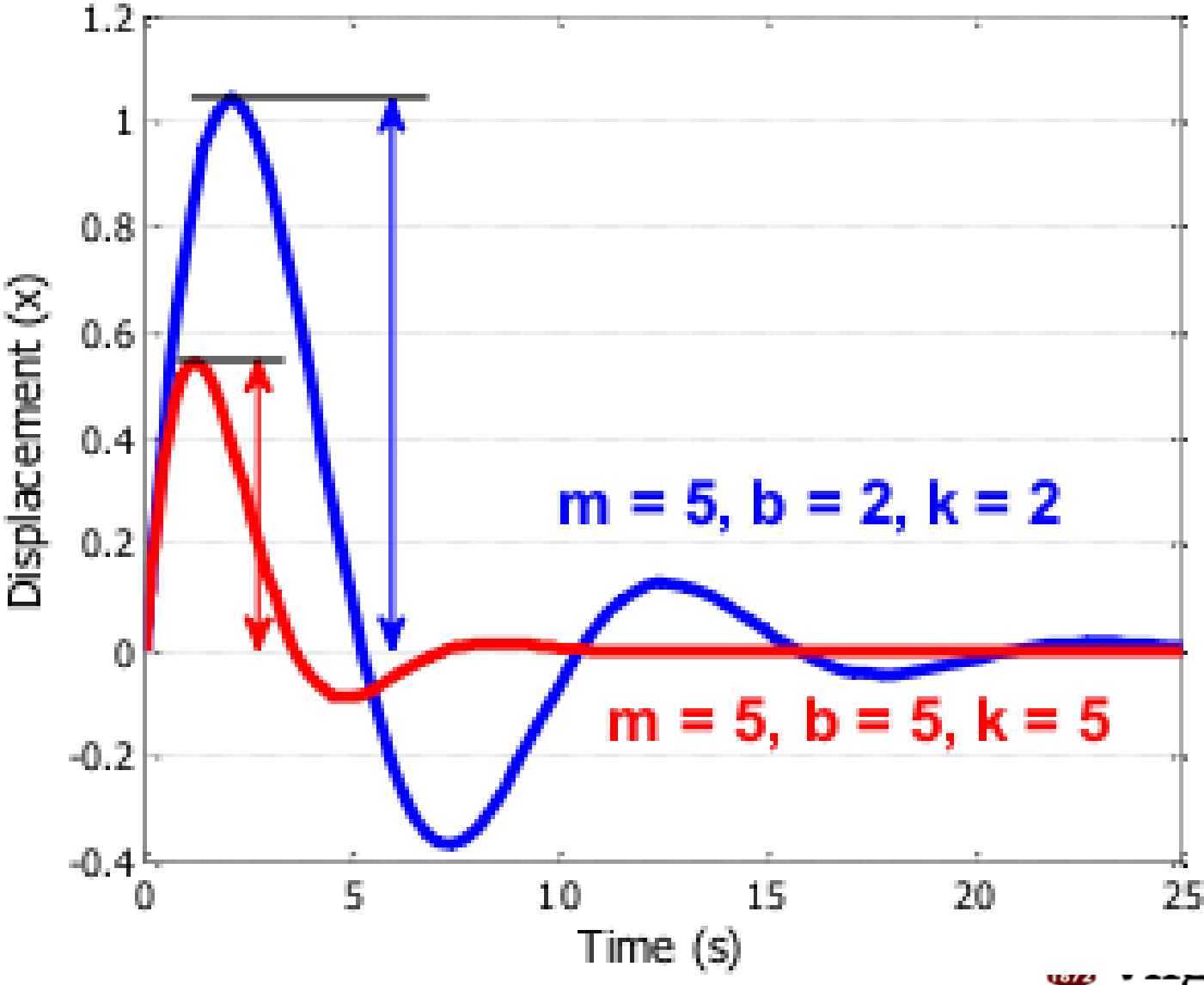
Each choice of parameters defines a solution, and let us suppose that the quantity of interest is the maximum deflection **xmax** that occurs for each solution.

We may wish to investigate the influence of b and k on this quantity, leaving m fixed and f zero.

So our computation might involve creating a plot of $xmax(b, k)$.

VirginiaTech

ODE: Each Solution has a Maximum Value



ODE: A Parameterized Problem

Evaluating the implicit function $xmax(b, k)$ requires selecting a pair of values for the parameters b and k, solving the ODE over a fixed time range, and determining the maximum value of x that is observed. Each point in our graph will cost us a significant amount of work.

On the other hand, it is clear that each evaluation is completely independent, and can be carried out in parallel. Moreover, if we use a few shortcuts in MATLAB, the whole operation becomes quite straightforward!

VirginiaTech

ODE: The Parallel Code

```
m = 5.0;
bVals = 0.1 : 0.05 : 5;
kVals = 1.5 : 0.05 : 5;

[ bGrid, kGrid ] = meshgrid ( bVals, kVals );

peakVals = nan ( size ( kGrid ) );

tic;

parfor ij = 1 : numel(kGrid)

  [ T, Y ] = ode45 ( @(t,y) ode_system ( t, y, m, bGrid(ij), ...
                                         kGrid(ij) ), [0, 25], [0, 1] );

  peakVals(ij) = max ( Y(:,1) );

end

toc;
```

ODE: PARPOOL or BATCH Execution

```
  pool_obj = parpool('local', 4);
  ode_sweep_parfor
  delete(pool_obj)
  ode_sweep_display
```

```
  job = batch ( ...
    'ode_sweep_script', ...
    'Profile', 'local', ...
    'AttachedFiles', {'ode_system.m'}, ...
    'pool', 4 );
  wait ( job );
  load ( job );
  ode_sweep_display
  delete ( job ) % destroy prior to R2012a
```

VirginiaTech

ODE: Display the Results

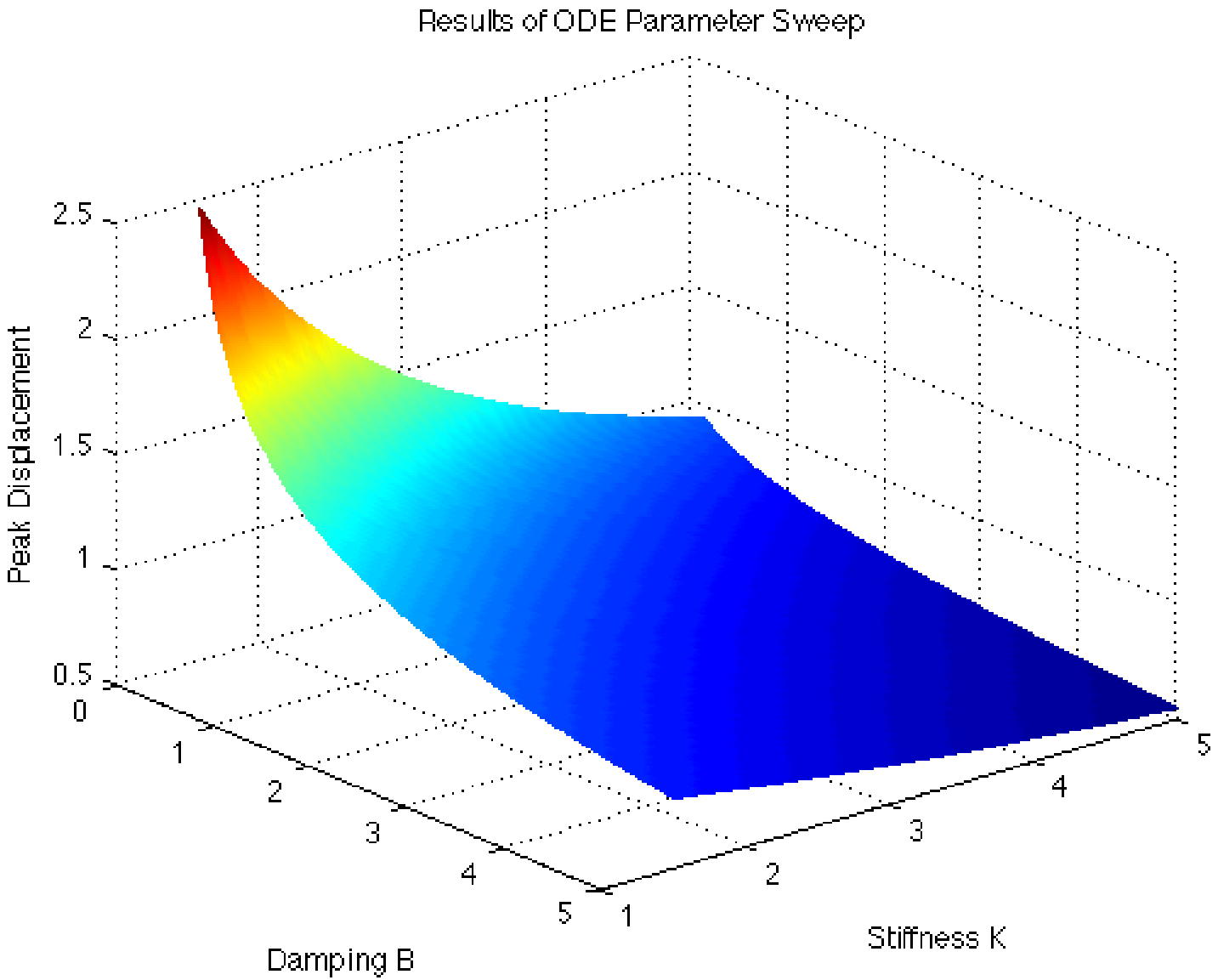
```
%
%  Display the results.
%
   figure;

   surf ( bVals, kVals, peakVals, 'EdgeColor', ...
                'Interp', 'FaceColor', 'Interp' );

   title ( 'Results of ODE Parameter Sweep' )
   xlabel ( 'Damping B' );
   ylabel ( 'Stiffness K' );
   zlabel ( 'Peak Displacement' );
   view ( 50, 30 )
```

VirginiaTech

ODE: A Parameterized Problem



aTech

ODE: A Very Loosely Coupled Calculation

In the PRIME program, the **parfor** loop was invoked in the outer loop; in each iteration there is a reasonable workload/.

In the ODE parameter sweep, we have several thousand IVP's to solve, but we could solve them in any order, on various computers, or any way we wanted to. All that was important was that when the computations were completed, every value $xmax(b, x)$ had been computed.

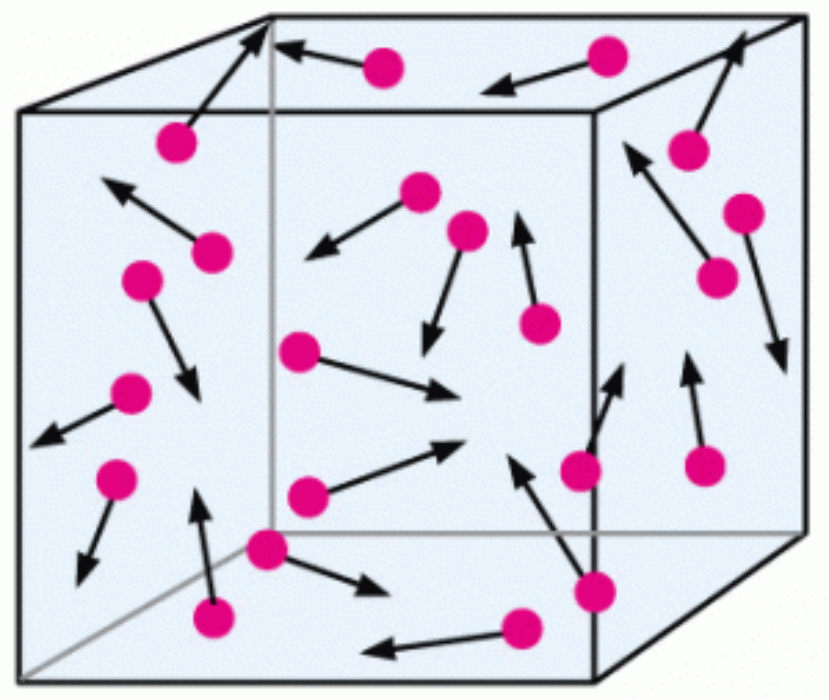
This kind of loosely-coupled problem can be treated as a *task computing* problem, wherein MATLAB can treat this problem as a collection of many little tasks to be computed in an arbitrary fashion and assembled at the end.

VirginiaTech

- Introduction
- FMINCON Example
- Executing a PARFOR Program
- PRIME Example
- Classification of variables
- ODE_SWEEP Example
- **MD Example**
- Conclusion

VirginiaTech

MD: A Molecular Dynamics Simulation



Compute the positions and velocities of **N** particles at a sequence of times. The particles exert a weak attractive force on each other.

VirginiaTech

MD: The Molecular Dynamics Example

The MD program runs a simple molecular dynamics simulation.

There are **N** molecules being simulated.

The program runs a long time; a parallel version would run faster.

There are many **for** loops in the program that we might replace by **parfor**, but it is a mistake to try to parallelize everything!

MATLAB has a **profile** command that can report where the CPU time was spent - which is where we should try to parallelize.

VirginiaTech

MD: Profile the Sequential Code

```
>> profile on
>> md
>> profile viewer
```

Step	Potential Energy	Kinetic Energy	(P+K-E0)/E0 Energy Error
1	498108.113974	0.000000	0.000000e+00
2	498108.113974	0.000009	1.794265e-11
...	...	...	...
9	498108.111972	0.002011	1.794078e-11
10	498108.111400	0.002583	1.793996e-11

```
CPU time  = 415.740000 seconds.
Wall time = 378.828021 seconds.
```

VirginiaTech

MD: Where is Execution Time Spent?

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
md	1	415.847 s	0.096 s	
compute	11	415.459 s	410.703 s	
repmat	11000	4.755 s	4.755 s	
timestamp	2	0.267 s	0.108 s	
datestr	2	0.130 s	0.040 s	
timefun/private/formatdate	2	0.084 s	0.084 s	
update	10	0.019 s	0.019 s	
datevec	2	0.017 s	0.017 s	
now	2	0.013 s	0.001 s	
datenum	4	0.012 s	0.012 s	
datestr>getdateform	2	0.005 s	0.005 s	
initialize	1	0.005 s	0.005 s	
etime	2	0.002 s	0.002 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead res
the process of profiling.

MD: The COMPUTE Function - nested j-loop

```
function [ f, pot, kin ] = compute ( np, nd, pos, vel, mass )

  f = zeros ( nd, np );
  pot = 0.0;

  for i = 1 : np
    for j = 1 : np
      if ( i ~= j )
        rij(1:nd) = pos(1:nd, i) - pos(1:nd, j);
        d = sqrt ( sum ( rij(1:nd).^2 ) );
        d2 = min ( d, pi / 2.0 );
        pot = pot + 0.5 * sin ( d2 ) * sin ( d2 );
        f(1:nd,i) = f(1:nd,i) - rij(1:nd) * sin ( 2.0 * d2 ) / d;
      end
    end
  end

  kin = 0.5 * mass * sum ( vel(1:nd,1:np).^2 );

  return
end
```

VirginiaTech

MD: The COMPUTE Function - vectorized j loop

```
function [ f, pot, kin ] = compute ( np, nd, pos, vel, mass )

  f   = zeros ( nd, np );
  pot = 0.0;
  pi2 = pi / 2.0;

  parfor i = 1 : np
    Ri  = pos - repmat ( pos( :, i ), 1, np );% array of vectors t
    D   = sqrt ( sum ( Ri.^2 ) );              % array of distances
    Ri  = Ri( :, ( D > 0.0 ) );
    D   = D( D > 0.0 );                        % save only pos valu
    D2  = D .* ( D <= pi2 ) + pi2*( D > pi2 );% truncate the poten
    pot = pot + 0.5 * sum ( sin ( D2 ).^2 );   % accumulate pot. en
    f( :, i) = Ri * ( sin ( 2*D2 ) ./ D )';    % force on particle
  end
%
%  Compute kinetic energy.
%
  kin = 0.5 * mass * sum ( diag ( vel' * vel ) );

  return
end
```

VirginiaTech

MD: Can We Use PARFOR?

The **compute** function fills the force vector **f(i)** using a **for** loop.

Iteration **i** computes the force on particle **i**, determining the distance to each particle **j**, squaring, truncating, taking the sine.

The computation for each particle is "independent"; nothing computed in one iteration is needed by, nor affects, the computation in another iteration. We could compute each value on a separate worker, at the same time.

The MATLAB command **parfor** will distribute the iterations of this loop across the available workers.

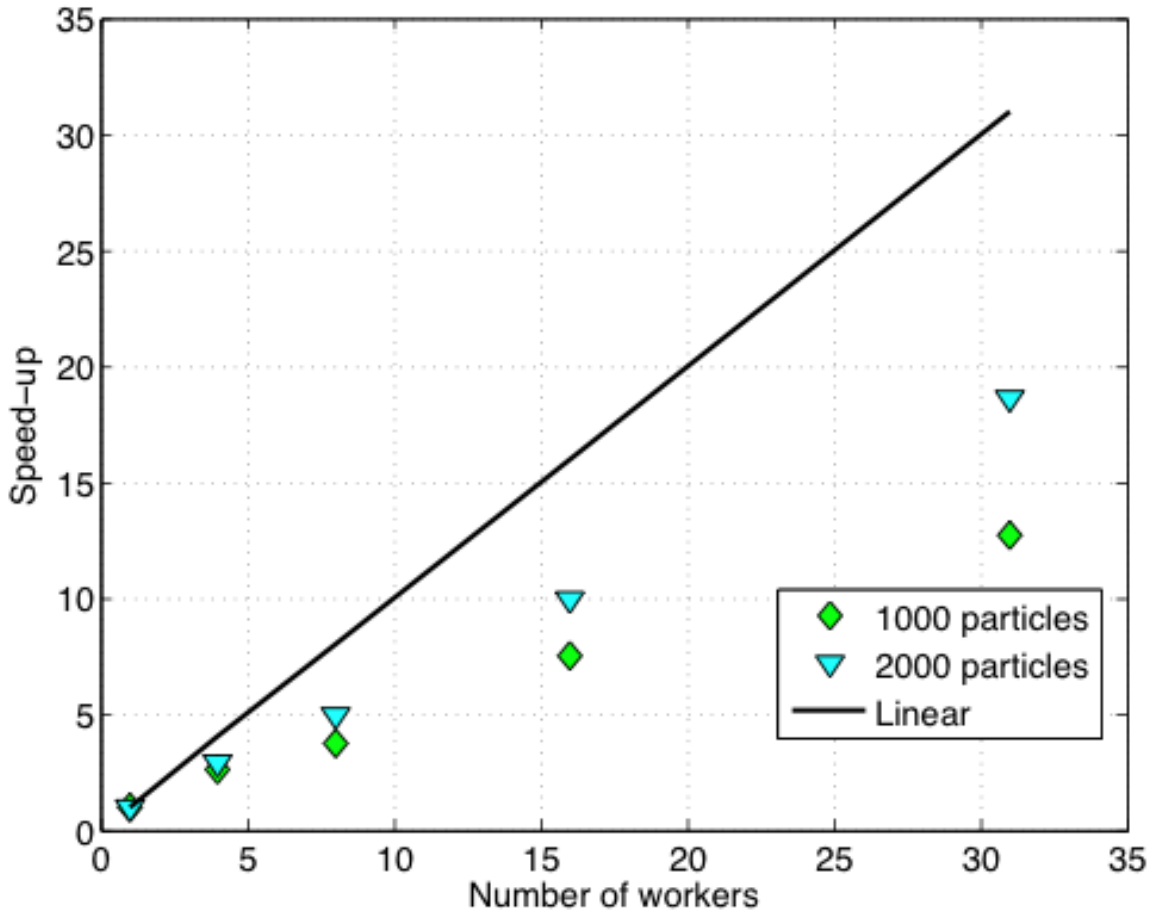
Tricky question: Could we parallelize the **j** loop instead?

Tricky question: Could we parallelize **both** loops?

VirginiaTech

MD: Speedup

Replacing **"for i"** by **"parfor i"**, here is our speedup:



virginiaTech

MD: Speedup

Parallel execution gives a huge improvement in this example.

There is some overhead in starting up the parallel process, and in transferring data to and from the workers each time a **parfor** loop is encountered. So we should not simply try to replace every **for** loop with **parfor**.

That's why we first searched for the function that was using most of the execution time.

The **parfor** command is the simplest way to make a parallel program, but in the next lecture we will see some alternatives.

VirginiaTech

MD: PARFOR is Particular

We were only able to parallelize the loop because the iterations were independent, that is, the results did not depend on the order in which the iterations were carried out.

In fact, to use MATLAB' **parfor** in this case requires some extra conditions, which are discussed in the PCT User's Guide. Briefly, **parfor** is usable when vectors and arrays that are modified in the calculation can be divided up into distinct slices, so that each slice is only needed for one iteration.

This is a stronger requirement than independence of order!

Trick question: Why was the scalar value **POT** acceptable?

VirginiaTech

- Introduction
- FMINCON Example
- Executing a PARFOR Program
- PRIME Example
- Classification of variables
- ODE_SWEEP Example
- MD Example
- **Conclusion**

CONCLUSION: Summary of Examples

In the **FMINCON** example, all we had to do to take advantage of parallelism was set an option (and *possibly* make sure some workers were available).

By timing the PRIME example, we saw that it is inefficient to work on small problems, or with only a few processors.

In the ODE_SWEEP example, the loop we modified was not a small internal loop, but a big "outer" loop that defined the whole calculation.

In the MD example, we did a profile first to identify where the work was.

VirginiaTech

CONCLUSION: Summary of Examples

We only briefly mentioned the limitations of the **parfor** statement.

You can look in the User's Guide for some more information on when you are allowed to turn a **for** loop into a **parfor** loop. It's not as simple as just knowing that the loop iterations are independent. MATLAB has concerns about data usage as well.

MATLABs built in program editor (`mlint`) knows all about the rules for using **parfor**. You can experiment by changing a **for** to **parfor**, and the editor will immediately complain to you if there is a reason that MATLAB will not accept a **parfor** version of the loop.

VirginiaTech

CONCLUSION: Desktop Experiments

Virginia Tech has a limited number of concurrent MATLAB licenses, and including the `Parallel Computing Toolbox`.

Since Fall 2011, the PCT is included with the student license.

Run `ver` in the MATLAB Command Window to see what licenses you have available.

If you don't have a multicore machine, you won't see any speedup, but you may still be able to run some 'parallel'programs.

VirginiaTech

CONCLUSION: Desktop-to-Ithaca Submission

- If you want to work with parallel MATLAB on ARC resources, you must first get an account. Go to

  `http://www.arc.vt.edu/account`

  Log in (PID and password), select the systems you want to work with and MATLAB in the Software section, and submit.

- Steps to set up submission from your desktop include:
  1. Download and add some files to your MATLAB directory
  2. Run a script to create a new profile on your desktop.

  A new cluster profile (e.g. `ithaca_R2015a`) will be created that can be used in `batch()`.
  These steps are described in detail at:

  `http://www.arc.vt.edu/matlabremote`

VirginiaTech

CONCLUSION: VT MATHWORKS LISTSERV

There is a local LISTSERV for people interested in MATLAB on the Virginia Tech campus. We try **not** to post messages here unless we really consider them of importance!

Important messages include information about workshops, special MATHWORKS events, and other issues affecting MATLAB users.

To subscribe to the MATHWORKS listserver, send a blank email to:

`mathworks-g+subscribe@vt.edu`

The subject and body of the message should both be empty.

VirginiaTech

CONCLUSION: Where is it?

- MATLAB Parallel Computing Toolbox Product Documentation
  http://www.mathworks.com/help/toolbox/distcomp/
- Gaurav Sharma, Jos Martin,
  MATLAB: A Language for Parallel Computing, International Journal of Parallel Programming,
  Volume 37, Number 1, pages 3-36, February 2009.
- An ADOBE PDF with these notes, along with a zipped-folder containing the MATLAB codes can be downloaded from the VT-arc web-site at

  `http://www.arc.vt.edu/matlab#resources`

VirginiaTech

THE END

Please complete the evaluation form

Thanks

VirginiaTech