



# MATLAB HPC Mentors June 25, 2020

Host: Nicholas Ide Guest Speakers: Ben Tordoff & Oli Tissot, Parallel Computing Development





#### **Quick Updates**

- No meeting in July or August enjoy the summer, stay healthy!
- If you have issues with recent SSH and MATLAB Parallel Server, let me know.
  - Have been a few reports
  - Members of the community have found temporary solutions



## **Distributed Arrays:**

# techniques and best practices for handling very large calculations



# Ben Tordoff & Oli Tissot, Parallel Computing Development



#### Agenda

What are distributed arrays?

- What can you do with distributed arrays?
- Building a distributed array
- Advanced manoeuvres
- Debugging methods and tips



#### Remote arrays in MATLAB

Rule: take the calculation to where the data is

Normal array – calculation happens in main memory:



x = rand(...)

 $x_norm = (x - mean(x)) . / std(x)$ 



#### Remote arrays in MATLAB

Rule: take the calculation to where the data is

gpuArray – all calculation happens on the GPU:

x = gpuArray(...)

x norm = (x - mean(x)) . / std(x)

distributed – calculation is spread across the memory of a cluster:

x = distributed(...)
x\_norm = (x - mean(x)) ./ std(x)

tall – calculation is performed by stepping through files:

$$x = tall(...)$$

$$x_norm = (x - mean(x)) . / std(x)$$



#### When should I reach for distributed?



If your data fits in memory, just use MATLAB normally

 If it won't fit on one machine, maybe it can be split across the combined memory of a cluster of machines? Use distributed arrays

 If it won't fit in the combined memory of a cluster of machines then use tall arrays



#### Using distributed arrays



- Use the memory of multiple machines as though it was your own
- Client sees a "normal" MATLAB variable
- Work happens on cluster



#### **Distributed Algorithms**

>> m = mean(d);





#### Agenda

- What are distributed arrays?
- What can you do with distributed arrays?
- Building a distributed array
- Advanced manoeuvres
- Debugging methods and tips



# What can you do with distributed arrays? Distributed array functions

MathMaulton			Help Center	Search Support - Q			Q							
	Mathworks <sup>®</sup>	Canada Cumant					Documentation	Examples	More -	Videos	Answers			
	Documentation       Examples       Functions       Videos       Answers         Run MATLAB Functions with Distributed A         Hundreds of functions in MATLAB <sup>®</sup> and other toolboxes are enhanced s         D = distributed(gallery('lehmer',n));         e = eig(D);         If any of the input arguments to these d[stributed-enabled functions is a MATLAB data is more appropriate (for example, nume1).         Distributed arrays are well suited for large mathematical computations, a arrays for big data processing. For more information on distributing arras         Check Distributed Array Support in Functions	Tays tistributed array, their output arrays are distributed, unless return uch as large problems of linear algebra. You can also use distributed is, see Distributing Arrays to Parallel Workers.	R2C	MATLAB Image Processing Toolbox Statistics and Machine Learning Toolbox Extended Capability Tall Arrays C/C++ Code Generation GPU Code Generation GPU Arrays Jistributed Arrays	316 354 166 381 518	•	struct2cell Operators and El Arithmetic Operatio +   sum   cumsum   -   diff   .*   *	ementary Opons Additi Sum o Cumu Subtra Differ Multip Matrix Produ Cumu	ert structure to perations on of array eleme ulative sum action ences and ap plication c multiplication uct of array ele ulative product array division	ents proximate de ments	rivatives		^	
	If a MATLAB function has distributed array support, you can consult add <b>Distributed Arrays</b> in the <b>Extended Capabilities</b> section at the end of <b>Tip</b> For a filtered list of all MATLAB functions that support distributed You can browse functions that support distributed arrays from all MathW Alternatively, you can filter by product. On the <b>Help</b> bar, click <b>Functions</b> MATLAB. At the bottom of the left pane, select <b>Distributed Arrays</b> . If you the <b>Distributed Arrays</b> filter is not available	tional distributed array usage information on its function page. She function page. arrays, lee Function List (Distributed Arrays). orks® products at the following link: All Functions List (Distributed In the function list, browse the left pane to select a product, for u select a product that does not have distributed-enabled function	d Arrays example ons, ther	s). e, 1	Ext – I	e In	nsive cludes	sup mos	port t line	: ar al	lgebi	ra		

- Scale up mathematical operations

#### Run MATLAB functions with distributed arrays



#### What can you do with distributed arrays?

- <u>>500 functions supported</u> (as of R2020a)
- support for dense and sparse linear algebra
- support for numerics (double, single, logical, etc.)
- support for datetimes, durations, categoricals, tables, ...

focus is on data preparation and large system solve



#### What can you do with distributed arrays?

Code written for distributed arrays looks like normal MATLAB code

```
% Create / read distributed data
A = distributed(...)
% Same code as for in-memory
b = sum(A, 2);
x1 = A \setminus b; % direct solution
x^2 = pcq(A, b); % iterative solution
% Bring back from cluster
[x1,x2] = gather(x1, x2);
```



#### Processing quite big data Multiplication of 2 NxN matrices



>> C = A \* B

	Execution time (seconds)						
Ν	1 node, 16 workers	2 nodes, 32 workers	4 nodes, 64 workers				
8000	19	13	11				
16000	120	75	50				
20000	225	132	86				
25000	-	243	154				
30000	-	406	248				
35000	-	-	376				
45000	-	-	743				
50000	-	-	-				

Processor: Intel Xeon E5-class v2 16 cores, 60 GB RAM per compute node, 10 Gb Ethernet



#### Agenda

- What are distributed arrays?
- What can you do with distributed arrays?
- Building a distributed array
- Advanced manoeuvres
- Debugging methods and tips



#### Building a distributed array

There are four main ways to build a distributed array:

- 1. Create from in-memory data
- 2. Build functions
- 3. Read from datastore
- 4. Construct from local parts



#### Building a distributed array: from in-memory

```
>> x = gallery("poisson", 10000);
```

```
>> dx = distributed(x); % Data sent to workers
```

- All data is sent from client to workers
- Useful for debugging before scaling up
- Useful for data that is close to filling local memory (i.e. can be created but not operated on due to fill-in etc.)



#### Building a distributed array: build functions

- >> dx = distributed.ones(1e9,100);
- >> dx = distributed.rand(1e9,100);
- >> dx = distributed.speye(le9);
- >> dx = distributed.sprand(1e9,1e9,1e-8);
- ... etc.
- No data is sent from client to workers
- Useful for creating test data and examples



## Building a distributed array: build from datastore

Datastore:

- Simple interface for data in multiple files/folders
- Presents data a piece at a time
- Access pieces in serial (desktop) or in parallel (cluster)
- Back-ends for tabular text, images, databases and more
- Data always stacked vertically

>>	ds = dat	astore("	'data/*.csv")
>>	preview(	ds)	
	Rows	Cols	Vals
	1	1	4
	2	1	-1
	10001	1	-1
	•	•	:
	•	•	:



## Building a distributed array: build from datastore

Read datastore into distributed

- each worker reads its own part of the data
- data is distributed vertically (workers have blocks of rows)

```
>> dt = distributed(ds); % distributed table of [Rows, Cols, Vals]
>> d = sparse(dt.Rows, dt.Cols, dt.Vals);
>> size(d)
ans = 100000000 100000000
>> nnz(d)
ans = 499960000
```



#### Agenda

- What are distributed arrays?
- What can you do with distributed arrays?
- Building a distributed array
- Advanced manoeuvres
- Debugging methods and tips



#### Writing your own algorithms

#### SPMD let's you craft parallel algorithms:

- Inside SPMD distributed -> codistributed
- getLocalPart extracts the data for this worker
- Use gop to reduce across workers (binary tree)
- Use gcat to concatenate results across workers

```
>> d = distributed(...)
>> spmd
        S = sum(getLocalPart(d)),"all"); % Sum values on this worker
        totalSum = gop(@plus, S); % Add results across all workers
    end
```



#### Writing your own algorithms

#### SPMD let's you craft parallel algorithms:

- MPI-like labSend, labReceive, labSendReceive for low level communication
- labindex for working out which worker you are

```
>> d = distributed(...)
>> spmd
mydata = getLocalPart(d);
% Cycle data to the right (wrapping round on last worker)
prevWorker = mod(labindex-2, numlabs)+1; % worker to the left
nextWorker = mod(labindex, numlabs)+1; % worker to the right
mydata = labSendReceive(nextWorker, prevWorker, mydata)
end
```



#### Modifying the data distribution

#### SPMD also gives control over distribution:

- redistribute for changing the distribution of an existing array
- codistributorXX for creating new (co)distributed arrays

```
>> d = distributed.ones(le5); % Default is 1D distribution in dim 2
>> spmd
       % Switch to having whole rows per worker
       d2 = redistribute(d, codistributor1d(1));
       % Use a custom (uneven) distribution of the data
       partition = 1e4 * [1 2 3 4];
       d3 = redistribute(d, codistributor1d(1, partition));
       % Switch to block-cyclic
       d4 = redistribute(d, codistributor2dbc());
   end
```



#### Agenda

- What are distributed arrays?
- What can you do with distributed arrays?
- Building a distributed array
- Advanced manoeuvres

Debugging methods and tips



# Debugging

Some rules of thumb:

- Start with "local" pool and small data
  - faster iteration time
  - proves correctness
- Use pre-built algorithms if you can
- If writing your own, prefer higher-level gop, gcat over labSend/Receive
  - It's easy to create mismatched communications with labSend/Receive!
  - Using unique tags for each communication helps to spot stray messages
  - Keeping computation and communication code separate helps in debugging both
- Run for a few different sizes to understand how duration scales with size
- Think about where your data lives minimize transmission of files across networks



# Debugging

#### Use parallel profiler (mpiprofiler):

 Shows what code ran, how much data transferred and allows comparison between workers (spot uneven loading)

```
>> mpiprofiler on
```

- >> % Lots of parallel code ...
- >> mpiprofview

				Profiler				
ile <u>E</u> dit De <u>b</u> ug <u>W</u> indow <u>H</u> elp								
← → 🖄 👌 🖊								
lab 1								
Parallal Profile Summary								
Faraller Profile Summary Generated 22	-jun-2020 16	:10:08 usin	ig real time	9.				
Showing <mark>all functions</mark> called in w	orker 1							
Automatic Comparison Selection	Manua	l Compa	rison Se	election		No Plo	t	
Compare (max vs. min TotalTime)	Gotov	vorker: 1		▼ S	how Figures (all w	orkers): Plot Ti	me Histograms Per Worker Commun	aication
Compare (max vs. min CommTime)	Compar	e with: Nor	ie	-		Plot Co	ommunication Time P	er Worker
* Communication statistics are not available for	r ScaLAPACK	functions,	so data m	arked with **	* might be inaccu	rate.		
Function Name	Calls	Total	Self	Total	Self Comm	Total	Computation	Total Time Plot
		<u>Time</u>	<u>Time</u> *	Comm	Waiting Time	Inter-worker	<u>Time Ratio</u>	(dark band is self time and
				IIme		Data		orange band is sell waiting time
remoteBlockExecution	13	2.072 s	0.009 s	0.011 s	0 s	2.37 Mb	99.5%	
remoteBlockExecutionPlain	13	2.015 s	0.019 s	0.011 s	0 s	2.37 Mb	99.5%	
spmd_feval_fcn>get_f/body	5	1.858 s	0.015 s	0.011 s	0 s	2.37 Mb	99.4%	
distributedSpmdWrapper>iInnerWrapper	2	1.779 s	0.026 s	0.011 s	0 s	2.37 Mb	99.4%	
odistributed.mldivide		1.001.4	0.040 0		-			
	1	1.001 8	0.049 5	0.005 s	0 s	2.19 Mb	99.7%	
codistributed.hMldivideAndMrdivide	1	1.606 s	0.049 s	0.005 s	0s 0s	2.19 Mb 2.19 Mb	99.7% 99.7%	
codistributed.hMldivideAndMrdivide @codistributed/private/scalaLuSolve	1	1.606 s 1.503 s	0.049 s 0.015 s 1.293 s	0.005 s 0.005 s 0.005 s	Os Os Os	2.19 Mb 2.19 Mb 2.19 Mb	99.7% 99.7% 99.7%	
codistributed.hMldivideAndMrdivide @codistributed/private/scalaLuSolve codistributed.redistribute	1 1 1 3	1.606 s 1.503 s 0.182 s	0.015 s 1.293 s 0.010 s	0.005 s 0.005 s 0.005 s 0.003 s	OS OS OS OS	2.19 Mb 2.19 Mb 2.19 Mb 2.19 Mb	99.7% 99.7% 99.7% 98.4%	
codistributed.hMldivideAndMrdivide @codistributed/private/scalaLuSolve codistributed.redistribute Redistributor.redistribute	1 1 1 3 3	1.606 s 1.503 s 0.182 s 0.159 s	0.049 s 0.015 s 1.293 s 0.010 s 0.022 s	0.005 s 0.005 s 0.005 s 0.003 s 0.003 s	OS OS OS OS OS	2.19 Mb 2.19 Mb 2.19 Mb 2.19 Mb 2.19 Mb	99.7%           99.7%           99.7%           99.7%           99.4%           98.1%	
codistributed.hMldivideAndMrdivide @codistributed/private/scalaLuSolve codistributed.redistribute Redistributor.redistribute codistributed.sum	1 1 1 3 3 1	1.606 s 1.503 s 0.182 s 0.159 s 0.092 s	0.015 s 1.293 s 0.010 s 0.022 s 0.006 s	0.005 s 0.005 s 0.005 s 0.003 s 0.003 s 0.003 s	Os Os Os Os Os Os	2.19 Mb 2.19 Mb 2.19 Mb 2.19 Mb 2.19 Mb 2.19 Mb 180.16 Kb	99.7%           99.7%           99.7%           99.7%           99.7%           98.1%           93.7%	
codistributed.hMldivideAndMrdivide @codistributed/private/scalaLuSolve codistributed.redistribute Redistributor.redistribute codistributed.sum codistributed.pSumAndProd	1 1 3 3 1 1 1	1.606 s 1.503 s 0.182 s 0.159 s 0.092 s 0.086 s	0.015 s 1.293 s 0.010 s 0.022 s 0.006 s	0.005 s 0.005 s 0.005 s 0.003 s 0.003 s 0.006 s 0.006 s	Os Os Os Os Os Os Os	2.19 Mb 2.19 Mb 2.19 Mb 2.19 Mb 2.19 Mb 180.16 Kb 180.16 Kb	99.7%           99.7%           99.7%           98.4%           98.1%           93.7%           93.2%	

# Debugging

Use spmd or parfevalonall to interact with workers and see local state

```
>> spmd, d, end
Lab 1: This worker stores d(:,1:1667).
          LocalPart: [10000x1667 double]
      Codistributor: [1x1 codistributor1d]
Lab 2: This worker stores d(:,1668:3334).
          LocalPart: [10000x1667 double]
      Codistributor: [1x1 codistributor1d]
Lab 3:
   etc.
```

