GPU Computing with Matlab II

Shuxia Zhang

Supercomputing Institute

University of Mineesota

e-mail: szhang@msi.umn.edu or help@msi.umn.edu

Tel: 612-624-8858 (direct) , 612-626-0802(help)

Outline

Introduction – Parallel computing with Matlab
- PCT
- GPUs

How to use multiple GPUs on one node?
- how many GPUs available
- How to use them
- parfor
- spmd

Hands-on exercises

Parallel computing with MATLAB

- MATLAB is widely used for developing/prototyping algorithms
- The High Level Language and Integrated Development/ Visualization environment leads to productive code development
- Parallelizing computing - A large-scale computation broken down into hundreds or thousands of independent units of work, each of which will run concurrently.
- Compute times may be reduced
- Solve larger-scale probles
- Explore bigger range of parameter space

1. Use of Multiple GPUs

Start matlab

module load matlab

matlab -r "MaxNumCompThreads(1)" % to use one thread

matlab % to use n threads (n cores)

Find resources and set the parallel environment

```
>> c=findResource  % CPU or cores
>> n=gpuDevice  % GPU devices
>> matlabpool(min(c,12));
```

Two ways to use of multiple GPUs

parallel for (parfor)

spmd

Implicit : Multithreading in MATLAB

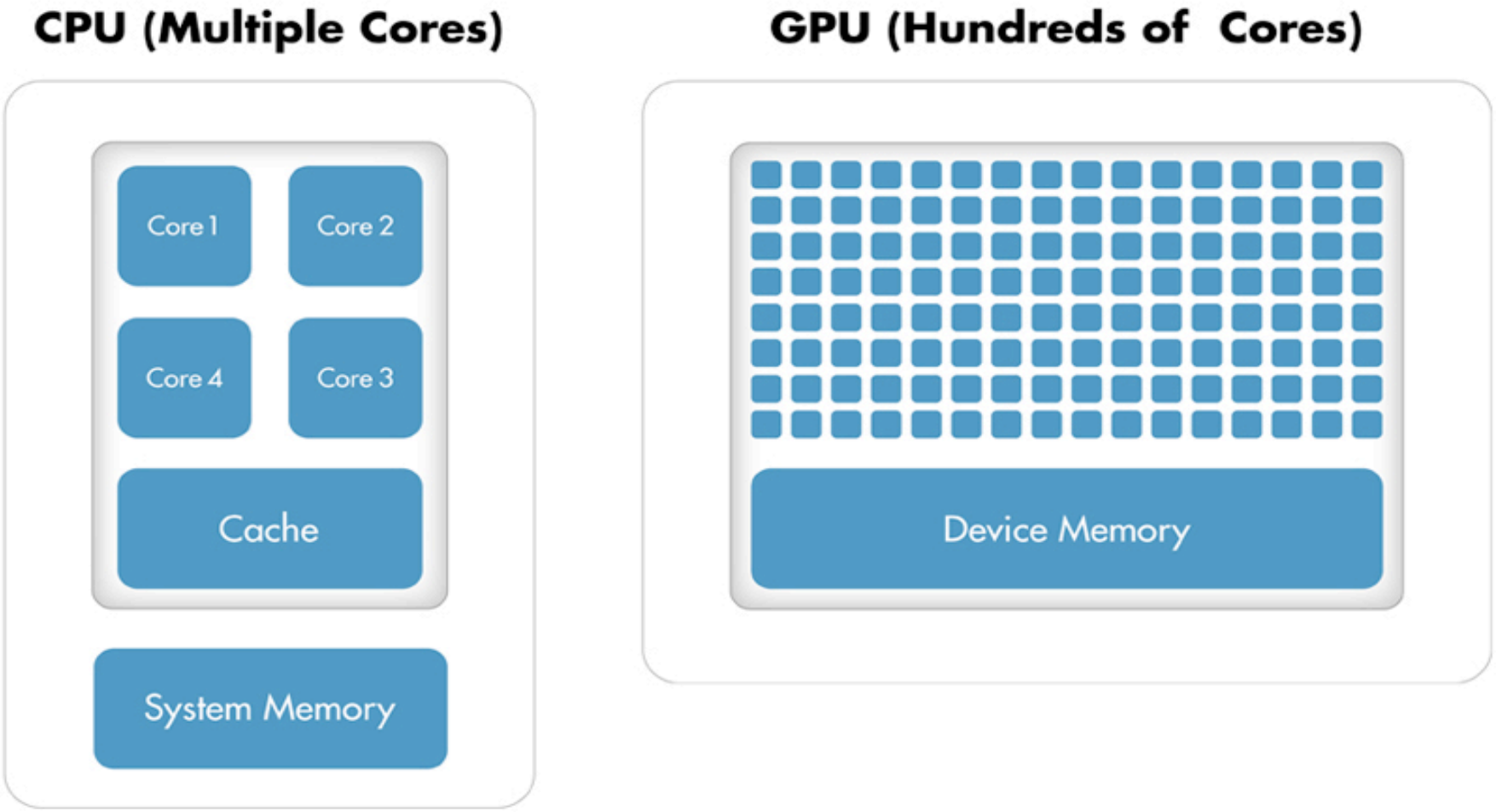
- MATLAB runs computations on multiple threads
- No changes to MATLAB code required
- Users can change behavior via preferences
- Maximum gain in element-wise operations and BLAS routines
- To see the performance improvements possible on your multi-core system, run the following demo:

```
>>  maxNumCompThreads
% set the number of threads to 1
>> maxNumCompThreads 1
```

Explicit multiprocessing

– The Parallel Computing Toolbox (PCT)
  in the mode of distributed memory, but only on one node.
– General purpose computing on GPU devices (GPGPU)
– MATLAB Distributed Computing Server (DCS),
  in the mode of distributed memory, across a series of
  computing nodes.
– Today we will focus on the use of PCT, throught which
  GPUs can be used. U of M does not buy the DCS license.

Parallel computing – why GPU?



Parallel Computing Toolbox

Hardware, SMP node with multicore processors and GPUs
PCT license is needed.
Matlab – R2013a or newer

Key Features
Provides twelve workers to execute the code.
Parallel for-loops (parfor)
Support for CUDA-enabled NVIDIA GPUs Ability
Distributed arrays and spmd (single-program-multiple-data)

Some PCT functions

```
%  help                  - Display help
%  dfeval                - Evaluate function
%  dfevalasync           - Evaluate function asynchronously
%  findResource          - Find available distributed computing
%  get                   - Return object properties
%  defaultParallelConfig - Control the default parallel
%  inspect               - Open Property Inspector
%  length                - Return length of object array
%  matlabpool            - Control an interactive session
%  parfor                - Parallel FOR-loop
%  pctconfig             - Configure settings for PCT
%  pmode                 - Control an interactive pmode session
%  taskFinish            - Task finish M-file
%  taskStartup           - Task startup M-file
```

Guidance for Use of GPUs

1. Use of gpuArray

```
>> N = 256; index1 = 1i*[0:N-1 0 1-N:-1];
>> index1 = gpuArray(index1);
>> index2=exp(index2); % calculation on GPU
>> plot(index2) % directly plot results
>> c=gather(index2);
```

gpuArray Characteristics	Description
classUnderlying	Class of the underlying data in the array
existsOnGPU	Indication if array exists on the GPU and is accessible
isreal	Indication if array data is real
length	Length of vector or largest array dimension
ndims	Number of dimensions in the array
size	Size of array dimensions

Guidance for Use of GPUs

2. Initialization directly on GPU

gpuArray.ones gpuArray.colon
gpuArray.zeros gpuArray.rand
gpuArray.inf gpuArray.randi
gpuArray.nan gpuArray.randn
gpuArray.true gpuArray.linspace
gpuArray.false gpuArray.logspace
gpuArray.eye

3. Run Built-In Functions on a GPU

A subset of the MATLAB built-in functions

>> help gpuArray/functionname

>> help gpuArray/lu

4. Run your own code: Only the first variable and arrays need the initialization using gpuArray

5. Run CUDA or PTX Code on GPU

6. Run MEX-Functions Containing CUDA Code

Use of GPU under PCT

One test case:

- Set up a benchmark for certain calculation (x = A\b)
  - for both single and double precision
  - problem size – fitting to memory
- Task 1: compare the performance
  - GPU vs CPU for single precision
  - GPU vs CPU for double precision
- Task 2: Effects of memory size on N GPUs
  - i.e., on different GPUs use different amount of memory and compare the performance
  - GPU vs CPU for single precision
  - GPU vs CPU for double precision

Introduction PCT
Start from a single GPU

```
g = gpuDevice;
maxMemory = 0.4*g.FreeMemory/1024^3;
maxSizeSingle = floor(sqrt(maxMemory*1024^3/4));
maxSizeDouble = floor(sqrt(maxMemory*1024^3/8)); step = 1024;
if maxSizeDouble/step >= 10
step = step*floor(maxSizeDouble/(5*step)); end
sizeSingle = 1024:step:maxSizeSingle/4;
sizeDouble = 1024:step:maxSizeDouble/2;
[cpu, gpu] = executeBenchmarks('single', sizeSingle,1);
results.sizeSingle = sizeSingle;
results.gflopsSingleCPU = cpu;
results.gflopsSingleGPU = gpu;
fig = figure; ax = axes('parent', fig);
plot(ax, results.sizeSingle, results.gflopsSingleGPU, '-x', results.sizeSingle, …
        results.gflopsSingleCPU, '-o')
grid on; legend('GPU', 'CPU', 'Location', 'NorthWest');
title(ax, 'Single-precision performance');
ylabel(ax, 'Gigaflops'); xlabel(ax, 'Matrix size'); drawnow;
```

Introduction PCT
Start from a single GPU

```
sizeDouble = 1024:step:maxSizeDouble/2;
ylabel(ax, 'Gigaflops'); xlabel(ax, 'Matrix size'); drawnow;
[cpu, gpu] = executeBenchmarks('double', sizeDouble,1);
results.sizeDouble = sizeDouble;
results.gflopsDoubleCPU = cpu;
results.gflopsDoubleGPU = gpu;
fig = figure; ax = axes('parent', fig);
plot(ax, results.sizeDouble, results.gflopsDoubleGPU, '-x', results.sizeDouble, …
results.gflopsDoubleCPU, '-o')
grid on; legend('GPU', 'CPU', 'Location', 'NorthWest');
title(ax, 'Double-precision performance');
ylabel(ax, 'Gigaflops'); xlabel(ax, 'Matrix size'); drawnow;
```

```
function [gflopsCPU, gflopsGPU] = executeBenchmarks(clz, sizes, igpu)
   fprintf(['Starting benchmarks with %d different %s-precision ' ...
       'matrices of sizes\nranging from %d-by-%d to %d-by-%d.\n'], ...
         length(sizes), clz, sizes(1), sizes(1), sizes(end), ...
         sizes(end));
   gflopsGPU = zeros(size(sizes));
   gflopsCPU = zeros(size(sizes));
   gd = gpuDevice(igpu);
   for i = 1:length(sizes)
      n = sizes(i);
      [A, b] = getData(n, clz);
      gflopsCPU(i) = benchFcn(A, b, @waitForCpu);
%        fprintf('Gigaflops on CPU: %f\n', gflopsCPU(i));
      A = gpuArray(A);
      b = gpuArray(b);
      gflopsGPU(i) = benchFcn(A, b, @() waitForGpu(gd));
      fprintf('Gigaflops on GPU: %f on Device %d \n', gflopsGPU(i),igpu);
   end
end
```

Use of parfor

Task parallel

- Same operation with different inputs
- No interdependencies between operations

Let us make another test for the same single GPU case, but on two GPUs with different amount of memory and plot the performance on different device:

- GPU vs CPU for single precision
- GPU vs CPU for double precision

In Parfor loop manner

```
findResource  % CPU or cores
n=gpuDeviceCount  % GPU devices
parfor id = 1:n
      g = gpuDevice(id);
      maxMemory = 0.3*g.FreeMemory/1024^3;
      if (id ==2)
      maxMemory = 0.5*g.FreeMemory/1024^3; end
      maxSizeSingle = floor(sqrt(maxMemory*1024^3/4));
      maxSizeDouble = floor(sqrt(maxMemory*1024^3/8)); step = 1024;
      if maxSizeDouble/step >= 10
      step = step*floor(maxSizeDouble/(5*step)); end
      sizeSingle = 1024:step:maxSizeSingle/5;
      Im=1:size(sizeSingle);
      [cpu, gpu] = executeBenchmarks('single', sizeSingle, id);
     fig = figure(id); ax = axes('parent', fig);
      plot(ax, sizeSingle, gpu, '-x', sizeSingle, cpu, '-o')
      grid on; legend('GPU', 'CPU', 'Location', 'NorthWest');
      title(ax, ['Single-precision performance on device' num2str(id)] );
     ylabel(ax, 'Gigaflops'); xlabel(ax, 'Matrix size'); drawnow;
end
```

In Parfor loop manner

```
findResource  % CPU or cores
 n=gpuDeviceCount  % GPU devices
                %Double precision calculation
 parfor id = 1:n
        g = gpuDevice(id);
        maxMemory = 0.3*g.FreeMemory/1024^3;
        if (id ==2)
        maxMemory = 0.5*g.FreeMemory/1024^3; end
        maxSizeDouble = floor(sqrt(maxMemory*1024^3/8)); step = 1024;
        if maxSizeDouble/step >= 10
        step = step*floor(maxSizeDouble/(5*step)); end
                sizeDouble = 1024:step:maxSizeDouble/4;
        [cpu, gpu] = executeBenchmarks('double', sizeDouble, id);

         fig = figure(id*2+1); ax = axes('parent', fig);
        plot(ax, sizeDouble, gpu, '-x', sizeDouble, cpu, '-o')
        grid on; legend('GPU', 'CPU', 'Location', 'NorthWest');
        title(ax, ['Double-precision performance on device' num2str(id)] );
         ylabel(ax, 'Gigaflops'); xlabel(ax, 'Matrix size'); drawnow;
end
```

Use of matlabpool and spmd

```
% matlabpool enables the parallel language features
% spmd - single program multiple data - allows interleaving of
% serial and parallel programming. The spmd environment is
% essentially equivalent to the pmode environment, but without
% the individual window for each worker.

    >> matlabpool open
    >>    spmd
    >>    ……<statements>
    >>    end
    >> matlabpool close

%Values generated in spmd region are saved as composite on client
```

Functions in spmd can use

%	labBarrier	- Block execution until all labs have reached this call
%	labBroadcast	- Send data to all labs or receive data sent to all lab
%	labindex	- Index of this lab
%	labProbe	- Test to see if messages are ready to be received
%	labReceive	- Receive data from another lab
%	labSend	- Send data to another specified lab
%	labSendReceive	- Simultaneously send and receive data
%	numlabs	- Total number of labs or processors

Collective Operations

The PCT provides the following collective operations

- gplus–Global addition
  - Example : p = gplus(s)
- gcat–Global concatenation
  - Example : c = gcat(s)
- gop–Global operation
  - Example : m = gop(@mean, s)

In SPMD manner

```
>> spmd;
id=labindex;
if (id < 3)
g=gpuDevice(id);
A=gpuArray.rand(1024,1024);
if (id ==1) B=fft(del2(A)); else B=fft(A); end
B=abs(B);
else
disp(' no GPUs')
end
end
```

```
>> whos
  Name      Size            Bytes  Class     Attributes
  A         1x2               697  Composite
  B         1x2               697  Composite
  g         1x2               697  Composite
  id        1x2               697  Composite
  p         1x2               697  Composite
```

```
>> mesh(cell2mat(B(1)));mesh(cell2mat(B(2)));
```

One of 2 CPUs
 one for FFT(A)
the other for FFT (laplace of A)
Calculate their absolute value

Can we add them togrther? How?

In SPMD manner

```
>> spmd;
id=labindex;
g=gpuDevice(id);
A=gpuArray.rand(1024,1024);
if (id ==1) B=fft(del2(A)); else B=fft(A); end
B=abs(B);
C =gop(@plus, B)
end

>> mesh(cell2mat(C(1)));mesh(cell2mat(C(2)));
```

Use of SPMD

Let us make another test for the same case set for parfor, but using spmd on two GPUs with different amount of memory and plot the performance on different device:

- GPU vs CPU for single precision
- GPU vs CPU for double precision

In SPMD manner

```
  findResource  % CPU or cores
  n=gpuDeviceCount  % GPU devices
cn=min(n,12);
 matlabpool(cn);
      spmd
        id=labindex
         g = gpuDevice(id);
         maxMemory = 0.3*g.FreeMemory/1024^3;
         if (id ==2)
         maxMemory = 0.5*g.FreeMemory/1024^3; end
         maxSizeSingle = floor(sqrt(maxMemory*1024^3/4));
         maxSizeDouble = floor(sqrt(maxMemory*1024^3/8)); step = 1024;
         if maxSizeDouble/step >= 10
         step = step*floor(maxSizeDouble/(5*step)); end
         sizeSingle = 1024:step:maxSizeSingle/5;
         Im=1:size(sizeSingle);
        [cpu, gpu] = executeBenchmarks('single', sizeSingle, id);
   end
for id=1:n
fig = figure(id); ax = axes('parent', fig);
         plot(ax, cell2mat(sizeSingle(id)),cell2mat(gpu(id)), '-x', cell2mat(sizeSingle(id)), cell2mat( cpu(id)), '-o')
         grid on; legend('GPU', 'CPU', 'Location', 'NorthWest');
         title(ax, ['Single-precision performance on device' num2str(id)] );
ylabel(ax, 'Gigaflops'); xlabel(ax, 'Matrix size'); drawnow;
end
```

%single precision calculation

Hands-on exercise:
Use of parfor or spmd

1. Login to cascade
   ssh -X cascade
2. Get a compute node to access gpu devices
   ssh -X cas001 # or
   ssh -X cas002 # or
   ssh –X cas003

3. Exercises
   Use of parfor - Set up a benchmark for certain calculation (x=fft(A)) for single precision and different array size on different GPUs subject to the available memory and compare the performance of GPU vs CPU.

Hands-on exercise:
Use of spmd

3. Exercises
   Use of SPMD - Set up a benchmark for certain calculation (x=fft(A)) for double precision and different array size on different GPUs subject to the available memory and compare the performance of GPU vs CPU.