MATLAB in HPC

GPU computing.

Anders Sjöström LUNARC
anders.sjostrom@lunarc.lu.se

What is needed?

- Matlab
- PCT
- GPU

Suitable problems

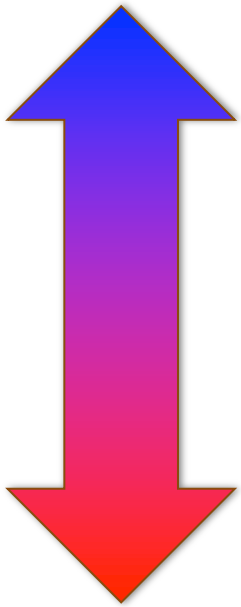
- Massively parallel tasks
- Computationally intensive tasks
- Tasks that have limited kernel size

Options

- Built-in functions
- Functions on array data
- Directly invoke CUDA-code

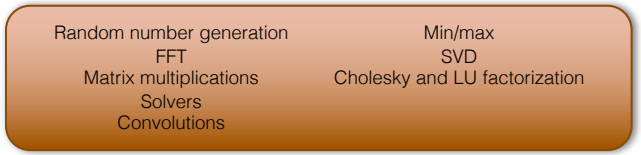
Control vs Effort

Level of control	Required effort
Minimal	Built-in functions
Some	Functions on array data
Extensive	Directly invoke CUDA code



Built-in functions

- Accelerate standard (highly parallel) functions
  - More than 200 MATLAB functions are already supported



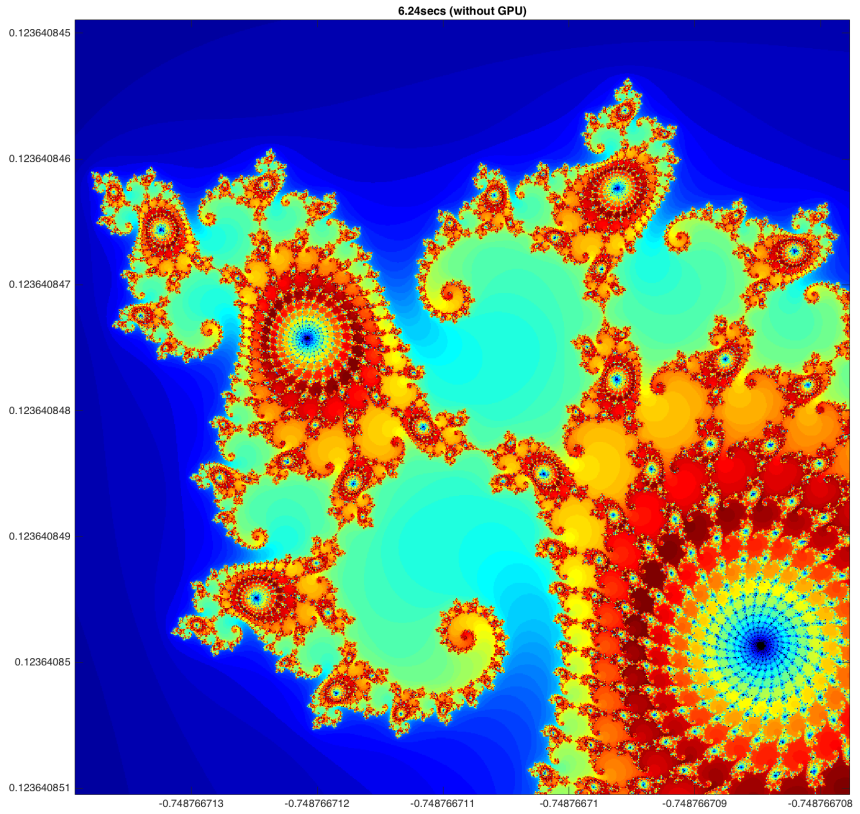
- Out of the box:
  - No additional effort for programming the GPU
- No accuracy for speed trade-off
  - Double floating-point precision computations

Example

CPU serial

```
maxIterations = 500;
gridSize=1000;
xlim = [-0.748766713922161, -0.748766707771757];
ylim = [ 0.123640844894862,  0.123640851045266];
t = tic();
x = linspace( xlim(1), xlim(2), gridSize );
y = linspace( ylim(1), ylim(2), gridSize );
[xGrid,yGrid] = meshgrid( x, y );
z0 = xGrid + 1i*yGrid;
count = ones( size(z0) );

% Calculate
z = z0;
for n = 0:maxIterations
    z = z.*z + z0;
    inside = abs( z )<=2;
    count = count + inside;
end
```




```
% show
count = log( count );
cpuTime = toc( t );

figure;
fig = gcf;
fig.Position = [200 200 600 600];
imagesc( x, y, count );
axis image
colormap( [jet();flipud( jet() );0 0 0] );
title( sprintf( '%1.2fsecs (without GPU)', cpuTime ) );
```

Example cont'd

Built-in functions

```
t = tic();
x = gpuArray.linspace( xlim(1), xlim(2), gridSize );
y = gpuArray.linspace( ylim(1), ylim(2), gridSize );
[xGrid,yGrid] = meshgrid( x, y );
z0 = complex( xGrid, yGrid );
count = ones( size(z0), 'gpuArray' );

% Calculate
z = z0;
for n = 0:maxIterations
    z = z.*z + z0;
    inside = abs( z )<=2;
    count = count + inside;
end
count = log( count );

% Show
count = gather( count ); % Fetch the data back from the GPU
naiveGPUTime = toc( t );
```

Example cont'd

Functions on array data

```
t = tic();
x = gpuArray.linspace( xlim(1), xlim(2), gridSize );
y = gpuArray.linspace( ylim(1), ylim(2), gridSize );
[xGrid,yGrid] = meshgrid( x, y );

% Calculate
count = arrayfun( @pctdemo_processMandelbrotElement, ...
                  xGrid, yGrid, maxIterations );

% Show
count = gather( count ); % Fetch the data back from the GPU
gpuArrayfunTime = toc( t );

function count = pctdemo_processMandelbrotElement(x0,y0,maxIterations)
z0 = complex(x0,y0);
z = z0;
count = 1;
while (count <= maxIterations) && (abs(z) <= 2)
    count = count + 1;
    z = z*z + z0;
end
count = log(count);
```

Example cont'd

PTX intermediate assembly file

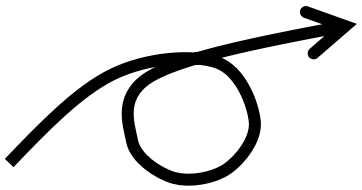
```
% Load the kernel
cudaFilename = 'pctdemo_processMandelbrotElement.cu';
ptxFilename = ['pctdemo_processMandelbrotElement.',parallel.gpu.ptxext];
kernel = parallel.gpu.CUDAKernel( ptxFilename, cudaFilename );
% Setup
t = tic();
x = gpuArray.linspace( xlim(1), xlim(2), gridSize );
y = gpuArray.linspace( ylim(1), ylim(2), gridSize );
[xGrid,yGrid] = meshgrid( x, y );
% Make sure we have sufficient blocks to cover all of the locations
numElements = numel( xGrid );
kernel.ThreadBlockSize = [kernel.MaxThreadsPerBlock,1,1];
kernel.GridSize = [ceil(numElements/kernel.MaxThreadsPerBlock),1];
% Call the kernel
count = zeros( size(xGrid), 'gpuArray' );
count = feval( kernel, count, xGrid, yGrid, maxIterations, numElements
);
% Show
count = gather( count ); % Fetch the data back from the GPU
gpuCUDAKernelTime = toc( t );
```

Example cont'd

PTX intermediate assembly file

```
__device__
unsigned int doIterations( double const realPart0,
                          double const imagPart0,
                          unsigned int const maxIters ) {
   // Initialize: z = z0
   double realPart = realPart0;
   double imagPart = imagPart0;
   unsigned int count = 0;
   // Loop until escape
   while ( ( count <= maxIters )
         && ((realPart*realPart + imagPart*imagPart) <= 4.0) ) {
      ++count;
      // Update: z = z*z + z0;
      double const oldRealPart = realPart;
      realPart = realPart*realPart - imagPart*imagPart + realPart0;
      imagPart = 2.0*oldRealPart*imagPart + imagPart0;
   }
   return count;
}
```

Nvcc –ptx code.cu



```
//
// Generated by NVIDIA NVVM Compiler
//
// Compiler Build ID: CL-19856038
// Cuda compilation tools, release 7.5, V7.5.17
// Based on LLVM 3.4svn
//

.version 4.3
.target sm_20
.address_size 64

        // .globl       _Z12doIterationsddj

.visible .func  (.param .b32 func_retval0) _Z12doIterationsddj(
        .param .b64 _Z12doIterationsddj_param_0,
        .param .b64 _Z12doIterationsddj_param_1,
        .param .b32 _Z12doIterationsddj_param_2
)
{
        .reg .pred      %p<3>;
        .reg .b32       %r<7>;
        .reg .f64       %fd<14>;


        ld.param.f64    %fd7, [_Z12doIterationsddj_param_0];
        ld.param.f64    %fd8, [_Z12doIterationsddj_param_1];
        ld.param.u32    %r4, [_Z12doIterationsddj_param_2];
        mov.u32 %r6, 0;
        mov.f64 %fd12, %fd8;
        mov.f64 %fd13, %fd7;

BB0_1:
        mov.f64 %fd2, %fd13;
        mov.f64 %fd1, %fd12;
        mul.f64 %fd3, %fd1, %fd1;
        mul.f64 %fd4, %fd2, %fd2;
        add.f64 %fd9, %fd4, %fd3;
        setp.gtu.f64    %p1, %fd9, 0d4010000000000000;
        @%p1 bra        BB0_3;

        add.s32 %r6, %r6, 1;
        sub.f64 %fd10, %fd4, %fd3;
        add.f64 %fd5, %fd10, %fd7;
        add.f64 %fd11, %fd2, %fd2;
        fma.rn.f64      %fd6, %fd11, %fd1, %fd8;
        setp.le.u32     %p2, %r6, %r4;
        mov.f64 %fd12, %fd6;
        mov.f64 %fd13, %fd5;
        @%p2 bra        BB0_1;

BB0_3:
        st.param.b32    [func_retval0+0], %r6;
        ret;
}
```

Example cont'd

cpuTime:	**1.6826 s**
naiveGPUTime:	**0.2775 s**
gpuArrayfunTime:	**0.0238 s**
Ptx-code:	**0.0112 s**

Source-code for the example can be found here:
https://se.mathworks.com/help/parallel-computing/examples/illustrating-three-approaches-to-gpu-computing-the-mandelbrot-set.html

Simple example

Matlab

```
k = parallel.gpu.CUDAKerne('test.ptx','test.cu');

result = feval(k,2,3)
```

```
//
// Generated by NVIDIA NVVM Compiler
//
// Compiler Build ID: CL-19856038
// Cuda compilation tools, release 7.5, V7.5.17
// Based on LLVM 3.4svn
//

.version 4.3
.target sm_20
.address_size 64

        // .globl       _Z4add1Pdd

.visible .entry _Z4add1Pdd(
        .param .u64 _Z4add1Pdd_param_0,
        .param .f64 _Z4add1Pdd_param_1
)
{
        .reg .f64       %fd<4>;
        .reg .b64       %rd<3>;


        ld.param.u64    %rd1, [_Z4add1Pdd_param_0];
        ld.param.f64    %fd1, [_Z4add1Pdd_param_1];
        cvta.to.global.u64      %rd2, %rd1;
        ldu.global.f64  %fd2, [%rd2];
        add.f64         %fd3, %fd2, %fd1;
        st.global.f64   [%rd2], %fd3;
        ret;
}
```

Kernel (test.cu)

```
__global__ void add1( double * pi, double c )
{
    *pi += c;
}
```

Multiple GPUs

```
parpool(2)


spmd
gd=gpuDevice;
idx=gd.Index;
disp(['Using GPU ',num2str(idx)]);
end


parfor ix = 1:10
  gd=gpuDevice;
  d(ix)=gd.Index;
end
```

```
Lab 1:
  Using GPU 1
Lab 2:
  Using GPU 2

>> d

d =

     2     2     2     2
1     1     1     2     2
1
```

LUND
UNIVERSITY