Parallel Computing for Science & Engineering

02/22/2018

Instructors:

Lars Koesterke, Charlie Dey

Home Work #0

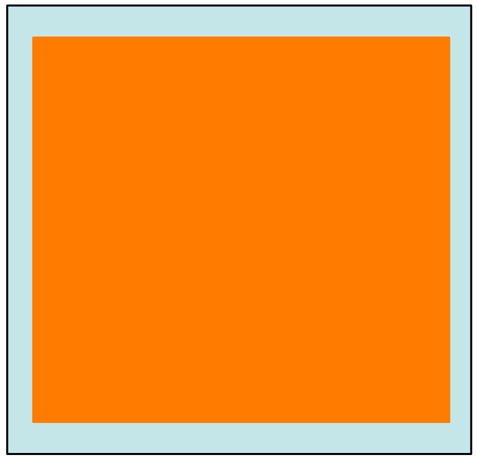
A common operation in scientific computing is digital convolution, by which each element in a multi-dimensional grid is replaced by a weighted sum of its neighbors. This has applications in graphics, in which such operations are done to both blur and sharpen images, and in numerical simulations, in which this may be a single step in a Poisson solver.
Write serial code for CPUs either in Fortran90 or in C. In the next homework you will port the code to the GPU and will compare timings on the CPU and the GPU.
Outline:
Define two 2D arrays (x, y) in your code. Initialize one array (x) with random numbers between 0 and 1. Derive the elements of the second array (y) from the first array by smoothing over the elements of the first array, using the constants a, b and c:

$$\begin{aligned} y(i,j) = {} & a \cdot (x(i-1,j-1) + x(i-1,j+1) + x(i+1,j-1) + x(i+1,j+1)) + \\ & b \cdot (x(i-1,j+0) + x(i+1,j+0) + x(i+0,j-1) + x(i+0,j+1)) + \\ & c \cdot x(i+0,j+0) \end{aligned}$$

Count the elements that are smaller than a threshold t in both arrays and print the number for both arrays. Collect timings for all ma jor steps in the code.

Data Layout

- Arrays x and y with (n+2)x(n+2) elements



- Inner elements (nxn) are going to be modified
- Provided boundaries "eliminate" if statements in the algorithm

Data Allocation

- Allocation on the heap:
  - More flexible
  - Stack size is limited
- Allocation in 1 step

```fortran
!***  Number of array elements in one direction
  integer                         :: n = 2**14
!***  Smothing constants
  real                            :: a = 0.05, &
                                     b = 0.1,  &
                                     c = 0.4

!***  Threshold
  real                            :: t = 0.1
!***  Input and output array
  real, dimension(:,:), allocatable :: &
                                     x, y

!***  Allocate input array
  allocate(x(0:n+1,0:n+1), stat=istat)
  allocate(y(0:n+1,0:n+1), stat=istat)
```

```c
int n,nbx,nby;
    float a,b,c,
        t,
        *x, *y;
    Timer timer;

/* n - number of elements in one direction */
    n = 1 << 14;  /* 2^14 == 16384 */

/* a,b,c - smoothing constants */
    a = 0.05;
    b = 0.1;
    c = 0.4;

/* t - threshold */
    t = 0.1;

/* allocate x */
    x = new float[(n+2)*(n+2)];

/* allocate y */
    y = new float[(n+2)*(n+2)];
```

Calls to Initialize, Smooth, Count

- All are subroutine /void function calls
- All information through subprogram parameters
- No global variables!

```fortran
!***  Initialize array x
call initialize(x, n)

!***  Derive second array from first array
call smooth(y, x, n, a, b, c)

!***  Count elements in first array
call count(x, n, t, nbx)

!***  Count elements in second array
call count(y, n, t, nby)
```

```c
 /* initialize x */
initialize(x, n);

/* smooth x into y */
smooth(y, x, n, a, b, c);

/* count elements in first array */
count(x, n, t, nbx);

/* count elements in second array */
count(y, n, t, nby);
```

Initialization and Smoothing

- In C: Index calculation “by hand”
- $\mathbf{x}_{i,j}$ `= x[ (n-1)*i + j ]`
- Fortran: **`random_number`** is an elemental function

```
void initialize( float* x, int n )
{
  int n2 = n+2;
  for (int i=0; i<n2; ++i)
    for (int j=0; j<n2; ++j)
      x[i*n2 + j] = random() / (float) RAND_MAX;
}
void smooth( float* y, float* x, int n,
             float a, float b, float c )
{
  int n2 = n+2;
  for (int i=1; i<=n; ++i)
    for (int j=1; j<=n; ++j)
      y[i*n2 + j] = a * (x[(i-1)*n2 + (j-1)] +
                         x[(i-1)*n2 + (j+1)] +
                         x[(i+1)*n2 + (j-1)] +
                         x[(i+1)*n2 + (j+1)])
                  + b * (x[i*n2+(j-1)] +
                         x[i*n2+(j+1)] +
                         x[(i-1)*n2 + j] +
                         x[(i+1)*n2 + j])
                  + c * x[i*n2 +j];
}
```

```
!***  Initialize with random numbers
subroutine initialize(x, n)
real, dimension(0:n+1,0:n+1) :: x
call random_number(x)
end subroutine

!***  Smooth data
subroutine smooth(y, x, n, a, b, c)
real, dimension(0:n+1,0:n+1) :: x, y
do j=1, n
  do i=1, n
    y(i,j) = a * (x(i-1,j-1) + x(i-1,j+1) + &
                  x(i+1,j-1) + x(i+1,j+1)) + &
             b * (x(i-0,j-1) + x(i-0,j+1) + &
                  x(i-1,j-0) + x(i+1,j+0)) + &
             c * x(i,j)
  enddo
enddo

end subroutine
```

Counting

- Count could have been a function!
- Return value is communicated through an argument

```
!***  Count elements below threshold
subroutine count(x, n, t, nbx)

real, dimension(0:n+1,0:n+1) :: x

nbx = 0

do j=1, n
  do i=1, n
    if (x(i,j) < t) then
      nbx = nbx + 1
    endif
  enddo
enddo

end subroutine
```

```
void count( float* x, int n, float t, int &nbx )
{
  nbx = 0;
  int n2 = n+2;
  for (int i=1; i <= n; ++i)
    for (int j=1; j <= n; ++j)
      if (x[i*n2 + j] < t)
        ++nbx;
}
```

Formatted Output (Fortran)

```
!***  Print number of elements below threshold
write (0,*)
write (0,'(a)')        'Summary'
write (0,'(a)')        '--------'
write (0,'(a,i14)')    'Number of elements in a row/column       :: ', n+2
write (0,'(a,i14)')    'Number of inner elements in a row/column :: ', n
write (0,'(a,i14)')    'Total number of elements                 :: ', (n+2)**2
write (0,'(a,i14)')    'Total number of inner elements           :: ', n**2
write (0,'(a,f14.5)')  'Memory (GB) used per array               :: ', real((n+2))**2 * 4. / (1024.
write (0,'(a,f14.2)')  'Threshold                                :: ', t
write (0,'(a,3(f4.2,1x))')  'Smoothing constants (a, b, c)            :: ', a, b, c
write (0,'(a,i14)')    'Number   of elements below threshold (X) :: ', nbx
write (0,'(a,es14.5)') 'Fraction of elements below threshold     :: ', real(nbx) / n**2
write (0,'(a,i14)')    'Number   of elements below threshold (Y) :: ', nby
write (0,'(a,es14.5)') 'Fraction of elements below threshold     :: ', real(nby) / n**2
```

Formatted Output (C++ & C)

```
std::cout << std::endl;
std::cout << "Summary" << std::endl;
std::cout << "-------" << std::endl;
std::cout << "Number of elements in a row/column          :: " << n+2 << std::endl;
std::cout << "Number of inner elements in a row/column    :: " << n << std::endl;
std::cout << "Total number of elements                    :: " << (n+2)*(n+2) << std::endl;
std::cout << "Total number of inner elements              :: " << n*n << std::endl;
std::cout << "Memory (GB) used per array                  :: " << (n+2)*(n+2)*sizeof(float) / (float
std::cout << "Threshold                                   :: " << t << std::endl;
std::cout << "Smoothing constants (a, b, c)               :: " << a << " " << b << " " << c << std::
std::cout << "Number   of elements below threshold (X)    :: " << nbx << std::endl;
std::cout << "Fraction of elements below threshold        :: " << nbx / (float)(n*n) << std::endl;
std::cout << "Number   of elements below threshold (Y)    :: " << nby << std::endl;
std::cout << "Fraction of elements below threshold        :: " << nby / (float)(n*n) << std::endl;
```

```
printf("\n");
printf("Summary\n");
printf("-------\n");
printf("Number of elements in a row/column        :: %d\n", n+2);
printf("Number of inner elements in a row/column :: %d\n", n);
printf("Total number of elements                  :: %d\n", (n+2)*(n+2));
printf("Total number of inner elements            :: %d\n", n*n);
printf("Memory (GB) used per array                :: %g\n", (n+2)*(n+2)*sizeof(float) / (float)(1024
printf("Threshold                                 :: %g\n", t);
printf("Smoothing constants (a, b, c)             :: %g %g %g\n", a,b,c);
printf("Number   of elements below threshold (X) :: %d\n", nbx);
printf("Fraction of elements below threshold      :: %g\n", nbx / (float)(n*n));
printf("Number   of elements below threshold (Y) :: %d\n", nby);
printf("Fraction of elements below threshold      :: %g\n", nby / (float)(n*n));
```

Sample Output

The output of your code may look like this:

```
Summary
-------
Number of elements in a row/column       :: 16386
Number of inner elements in a row/column :: 16384
Total number of elements                 :: 268500996
Total number of inner elements           :: 268435456
Memory (GB) used per array               :: 1.00024
Threshold                                :: 0.10
Smoothing constants (a, b, c)            :: 0.05 0.10 0.40
Number of elements below threshold (X)   :: 26847453
Fraction of elements below threshold     :: 1.00015E-01
Number of elements below threshold (Y)   :: 2950
Fraction of elements below threshold     :: 1.09896E-05

Action       :: time/s Time resolution = 1.0E-04
------
CPU: Alloc-X :: 0.000
CPU: Alloc-Y :: 0.000
CPU: Init-X  :: 3.904
CPU: Smooth  :: 1.434
CPU: Count-X :: 0.397
CPU: Count-Y :: 0.384
```

Timing

- We will be using timers left and right
- You can use a simple timer
- Our implementation uses a timer object. What is an object?
- How to declare an object?
- How to call an object?

```
!***  Timing information
class(cls_timer) :: timer

!***  Count elements in second array
call timer%start('CPU: Count-Y')
call count(y, n, t, nby)
call timer%stop
```

```
/* timer of class Timer */
Timer timer;

 /* count elements in second array */
timer.start("CPU: Count-Y");
count(y, n, t, nby);
timer.stop();
```

Timing

- We will be using timers left and right
- You can use a simple timer
- Our implementation uses a timer object. What is an object?
- How to declare an object?
- How to call an object?

```fortran
!***  Timing information
class(cls_timer) :: timer

!***  Count elements in second array
call timer%start('CPU: Count-Y')
call count(y, n, t, nby)
call timer%stop
```

```c
/* timer of class Timer */
Timer timer;

 /* count elements in second array */
timer.start("CPU: Count-Y");
count(y, n, t, nby);
timer.stop();
```

Please ignore the subsequent slides if you do not want to introduce a timer class.

Timing

- A class contains private data and public methods

```fortran
type, public :: cls_timer

private
integer                              :: n = 0
integer(8), dimension(2,m)           :: it
integer(8)                           :: itr
character(len=16), dimension(m)      :: c

contains

procedure, public :: reset
procedure, public :: start
procedure, public :: stop
procedure, public :: print

endtype cls_timer

contains

subroutine reset(this)

class(cls_timer) :: this
this%n  = 0
this%it = -1
this%c  = 'undef'

end subroutine

…
```

```cpp
class Timer
{
public:
  Timer(): n(0) { }
  void start(std::string label)
  {
    if (n < 20)
      { labels[n] = label; times[2*n] = clock(); }
    else { std::cerr << "No more timers, " << label
      << " will not be timed." << std::endl; }
  }

  void stop() { times[2*n+1] = clock(); n++;}
  void reset() { n=0; }
  void print();
private:
  std::string labels[20];
  float times[40];
  int n;
};
…
```

Most important method: Printing

```
subroutine print(this)

     class(cls_timer) :: this

     write (0,*)
     write (0,'(a,es7.1)') 'Action          ::   time/s     Time resolution = ', 1./real(this%itr
     write (0,'(a)')       '------'
     do i=1, this%n
       write (0,'(a16,a, f7.3)') this%c(i), ' :: ',               &
                            (real(this%it(2,i) - this%it(1,i))) / real(this%itr)
     enddo

     end subroutine
```

```
void Timer::print()
{
    std::cout << std::endl;
    std::cout << "Action          ::   time/s     Time resolution = " << 1.f/(float)CLOCKS_PER_SEC
    std::cout << "------" << std::endl;
    for (int i=0; i < n; ++i)
        std::cout << labels[i] << " :: " << (times[2*i+1] - times[2*i+0])/(float)CLOCKS_PER_SEC <<
}
```

TACC

Invoke printing by

- Invoking the method **`print`**

```
!***  Print timings
      call timer%print
```

```
/* Print timings */
timer.print();
```

- If you need another (probably nested) timer: create another object

```
!***  Timing information
class(cls_timer) :: timer, timer2

!***  Count elements in second array
call timer2%start('CPU: Count-XY')
  call timer%start('CPU: Count-X')
  call count(x, n, t, nbx)
  call timer%stop
  call timer%start('CPU: Count-Y')
  call count(y, n, t, nby)
  call timer%stop
call timer2%stop
```

```
/* timer of class Timer */
Timer timer timer2;

 /* count elements in second array */
timer2.start("CPU: Count-XY");
  timer.start("CPU: Count-X");
  Count(x, n, t, nbx);
  timer.stop();
  timer.start("CPU: Count-Y");
  count(y, n, t, nby);
  timer.stop();
timer2.stop();
```

Header files; Modules

- Fortran: Subroutines/Functions/Class declarations are put in Modules
  - Modules become available by use association
  - This implies also an order of compilation: Modules first, then code that uses the module
- C/C++: Header files are used to communicate the interface

```fortran
! Example for a module
Module my_mod
Contains
Subroutine do_this(x, y, z)
…
End subroutine
End module

Program main
Use my_mod
Call do_this(a,b,c)
End program
```

```cpp
/* Header files needed for HW0
   I/O
   timer
   strings */
#include <iostream>
#include <sys/time.h>
#include <string>
```

What a timer class should provide

- A method that deals with nested timers
- What else?