SCC 374C/394C: Parallel Computing for Science and Engineering

Assignment #0 — Serial (CPU) code

Due 11:59PM, Feb 7, 2017

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 **CPU**s either in Fortran90 or in C. In upcoming homework you will parallelize the code with OpenMP and MPI and will compare timings of the serial and parallel code versions.

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:

```
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-1,j+0) + x(i+1,j+0) + x(i+0,j-1) + x(i+0,j+1)) +
         c·x(i+0,j+0)
```

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

Instructions

- Write one main program in C (or C++) or Fortran. Use single precision *float/real* variables throughout the excercise.
- Write 3 functions returning *void* in C or 3 subroutines in Fortran, respectively.
- Initialize the constants *a=0.05*, *b=0.1* and *c=0.4* in the main program.
- Initialize the threshold *t=0.1* in the main program.
- Initialize the number of array elements in each direction to *n=16384+2*.
- Declare two arrays (x and y):
  - Allocate the arrays dynamically. Use *allocatable arrays* in Fortran and *malloc* in C.
  - Allocate the arrays such that the number of elements in each direction is n.

  - Make sure that each of your 2D arrays is contiguous in memory (Array allocated in one step).

- Subprogram to initialize an array with random numbers: *initialize*().
  - The subprogram takes two arguments: The array itself and the number of elements in one direction.
  - Initialize the whole array, i.e., all n^2 elements with random numbers between 0 and 1.
  - Fortran: Use the built-in subroutine *random_number*().
  - C: Use *random*() / *(float)RAND_MAX* from stdlib.h.

- Subprogram to smooth data: *smooth*().
  - The subprogram takes six arguments: input array, output array, the number of elements in one direction, and the three smoothing constants a, b and c .
  - Implement the smoothing algorithm presented above.
  - Note that the smoothing algorithm uses elements surrounding the actual position *(i,j)*. Calculate only those elements of the output array that are inner elements, i.e., that are **not** at the boundary of the 2D array. Therefore the process for the calculation of an element in the output matrix is identical for every matrix element. No *if* statements will be needed and a total $(n-2)^2$ elements of y are calculated.

- Subprogram to count all elements below a threshold: *count*().
  - The subprogram takes four arguments: array and the number of elements in one direction, the threshold, and the number of elements below the threshold.
  - Count the number of elements.
  - Count only the inner $(n-2)^2$ elements!
  - This subprogram will be called twice for x and y.

- Time all major code sections:
  - Allocation of the arrays. Time both allocations separately.
  - Time all calls to the 3 subprograms separately (a total of 4 calls).
  - Collect all timing information in the *Main Program*.
  - Print the timing information in a table at the end.
  - Fortran: Use the F90 timer *system_clock*().
    - *system_clock(i, j)* accepts 2 integer arguments.
    - The first one (i) is the count, the second one (j) is the count rate.
    - The time elapsed between 2 calls to *system_clock*() is:
      `t = (real(i2) - real(i1)) / real(j)`.
  - C: Use the C time *clock*() in time.h.
    - *clock*() takes no arguments. It uses a fixed count rate *CLOCKS_PER_SEC*.

    * The time elapsed between two calls to *clock*() is:
      `float t = (i2 - i1) / (float)CLOCKS_PER_SEC;`

- All "communication" between the *main program* and the *functions/subroutines* have to be done through subprogram parameters.
- Special instruction for Fortran programmers
  - Create a *module* that is being used by the *main program*.
  - Add all 3 *subroutines* to this module.
- Your main program should be structured like this:
  - Declaration of variables. Constants are preset.
  - Allocation of the arrays x and y.
  - Call of subprogram *initialize*() for array x.
  - Call of subprogram *smooth*() to derive y from x.
  - Call of subroutine *count*() twice for arrays x and y.
  - Print the total number of elements in a column/row of an array.
  - Print the total number of inner elements in a column/row of an array.
  - Print the total number of elements in an array.
  - Print the total number of inner elements in an array.
  - Print the number of bytes that are used by one array. Use appropriate units!
  - Print the number of elements below the threshold for both arrays.
  - Print the fraction of elements below the threshold and the threshold itself.
  - Print a table with the timing information (6 separate timings).

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
```