CSCI 373: Parallel Programming
Spring 2017
Prof. K. Sanft
Exam 1 (100 Points)

This is an exam, therefore, it must be completed independently. You may search for MPI syntax-related questions but you may not search for solutions or code for this or similar problems.

The due date is Friday, March 10, by 2:00pm. The Moodle link will not accept submissions after that time.

Disclaimer: this is an initial draft. Particular parameters may be specified later. The class will be notified if a new version is posted.

**Background:**

Consider x and y in the range [0, 1]. We discretize x and y into N points, where $x_0 = 0$ and $x_{N-1} = 1$ and $x_i = i/(N-1)$, and $y_0 = 0$ and $y_{N-1} = 1$ and $y_i = i/(N-1)$.

For example, if N = 6, then: $x_0 = 0$, $x_1=.2$, $x_2=.4$, $x_3=.6$, $x_4=.8$, $x_5=1.0$ and $y_0 = 0$, $y_1=.2$, $y_2=.4$, $y_3=.6$, $y_4=.8$, $y_5=1.0$.

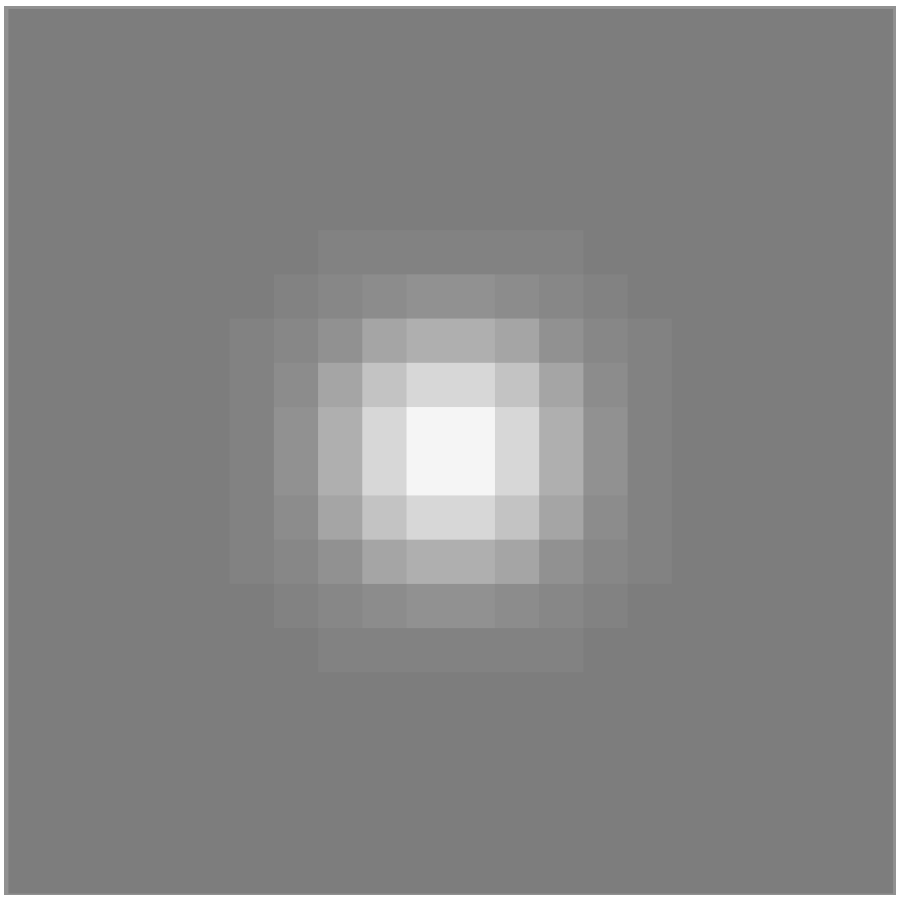
Consider a function $f_k(x, y)$ which we compute using the iterative formula:

$f_k(x_i, y_j) =$
$0.01*(f_{k-1}(x_{i-1}, y_j) + f_{k-1}(x_{i+1}, y_j) + f_{k-1}(x_i, y_{j-1}) + f_{k-1}(x, y_{j+1}) - 4*f_{k-1}(x_i, y_j)) + 2*f_{k-1}(x_i, y_j) - f_{k-2}(x_i, y_j)$

Note that to compute f at step k at point $x_i$, $y_j$, we need the value of f at step k-1 at point $x_i$, $y_j$ and its four "neighbors" and we need the value of f at step k-2 at point $x_i$, $y_j$.

We will let $f_k(0, y) = 0$, $f_k(1,y) = 0$ $f_k(x, 0) = 0$ and $f_k(x, 1) = 0$ for all k. That is, the outer boundary of our domain will always be 0 (for all k). Therefore, we need to compute $f_k(x_i, y_j)$for all combinations of i = 1, 2, …, N-2 and j = 1, 2, …, N-2.

Let $f_0(x_i, y_j) = f_1(x_i, y_j)$ = initialCondition($x_i$,$y_j$), where the function initialCondition can be found in initialCondition.c. Note there is a variable "sigma" that can be adjusted to modify the initial condition. With sigma=0.1, and N=20, the initial condition looks like this (enlarged from its 20x20 pixel dimensions):



**Problem:**

Consider using MPI to compute $f_2$, $f_3$, …$f_{end}$, where *end* is a variable in your program. You may assume that N % comm_sz == 0.

You can assume that comm_sz is even.

You can assume that N / comm_sz >= 4.

You will get the highest point value you achieve (if you complete the 100 point version successfully, you don't have to submit the 65 point version—but you may want to anyway, just in case.) Partial credit may be awarded. It is typically better to have an incomplete solution that compiles and runs rather than to have almost any version that does not compile. A program that crashes will likely receive very few points.

65 points:

- Solve to step *end*, where *end* is a variable in your program, in serial (pure C or in mpi with 1 process). It should work for values of end>=2 and N>=8.

- Write .pgm files for $f_1$, $f_2$, …, $f_{end}$. Name your files with consistent names of the form: output0001.pgm, output0002.pgm, … , output0010.pgm, output0011.pgm,… etc. Your output files must have that file name format for the command on the next bullet point to work. (You do not have to submit any .pgm files to Moodle.)
- Create a .gif animation from the files. This can be done on the lab machines with the command:
  ```
  convert –verbose output*.pgm  –resize 400x400 –loop 1 animation.gif
  ```
  Turn in the animated gif file.

75 points (a correct parallel implementation on 2 procs):

- Solve to step *end* in parallel with comm_sz=2.
- Each proc should store only its portion of the full 2D array. That is, each proc should store an (N/2)xN array, NOT the entire NxN array. You should allocate only enough memory for the current output time and the two previous and then you should reuse that memory for successive steps.
- Your code should create the pgm files as above. For writing the output, it is acceptable to gather the entire 2D array on proc 0 and have proc 0 write the files.

Up to 100 points (a correct and *efficient* parallel implementation):

- Solve to step *end* in parallel with comm_sz>=2.
- As above, each proc should not store more than its share of the total 2D arrays and should not store more than 3 time points worth of data.
- As above you should be able to output pgm files. It's fine
- Include a variable that determines whether or not to produce output files.
- Include code to time your code. Time your code on killdevil and produce a plot(s) to show the performance. DO NOT CREATE OUTPUT FILES ON KILLDEVIL.
- Efficiency (except for when writing output files) is key to receiving maximum points. Avoid extraneous tasks that slow down the computations if there are more efficient ways to implement them.
- A "fastest time" bonus will be awarded to the student whose solution runs (correctly) most efficiently. (Prof. Sanft will determine the parameters for that run and perform the timing experiments.)
    - For the fastest time bonus, you must use double precision and must not use "tricks" like exploiting the symmetry of the problem to solve just ¼ of the domain, etc. The "clever modifications" described below are allowed.

Bonus points may also be awarded for clever modifications (along with an explanation of the modification and the results). Bonus points will not be awarded unless a correct and (reasonably) efficient full solution is as described above provided. Some ideas for modifications include: use square partitioning of the domain (with comm_sz = a perfect square), send two rows of neighbor data so you can do two steps per communication, other ideas?