MPP coursework

Details and answers to common questions

epcc

THE UNIVERSITY OF EDINBURGH

Reusing this material

epcc

THE UNIVERSITY OF EDINBURGH

Edge Detection

- Represent pixels in an *M*x*N* image as a 2D array of values $image_{i,j}$ in the range 0 (black) to 255 (white)
  - *i* is in the range 1, 2, ..., *M*; *j* is in the range 1, 2. ..., *N*
  - simple to store as an ASCII PGM file
- Apply a simple edge detection algorithm – combine each pixel with its 4 nearest neighbours

$$edge_{i,j} = image_{i+1,j} + image_{1,j+1} + image_{i-1,j} + image_{i,j-1} - 4\,image_{i,j}$$

- What about the pixels at the edges since neighbours are outside the original image?
  - set the boundary conditions so all these values are 255 (white)

Image reconstruction

- We choose to reconstruct the image from the edges
  - turns out that this *is* possible given the way we compute the edges

- Algorithm
  - make an initial guess at the image values old_{ij}
  - update *many times* using this equation:

$$new_{i,j} = \frac{1}{4}\left(old_{i+1,j} + old_{1,j+1} + old_{i-1,j} + old_{i,j-1} - edge_{i,j}\right)$$

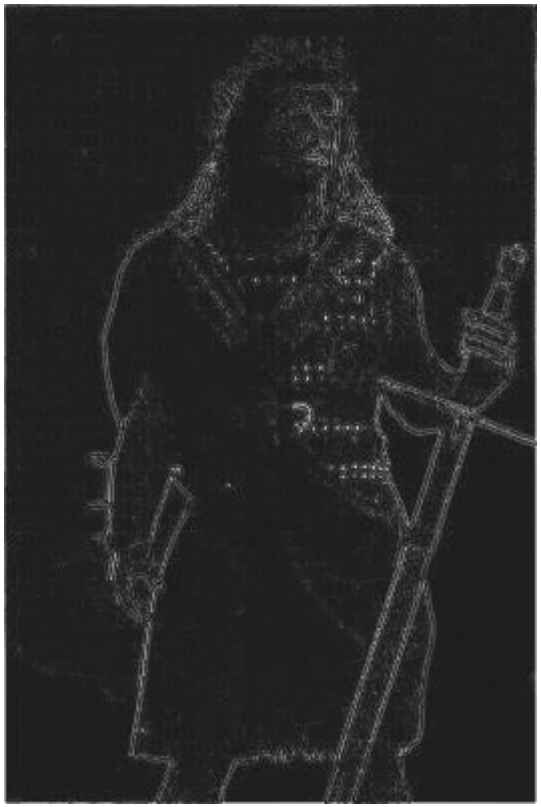
    - copy *new* back to *old*

  - repeat ...

Edge detection / image reconstruction

single pass

hundreds of iterations



Serial code

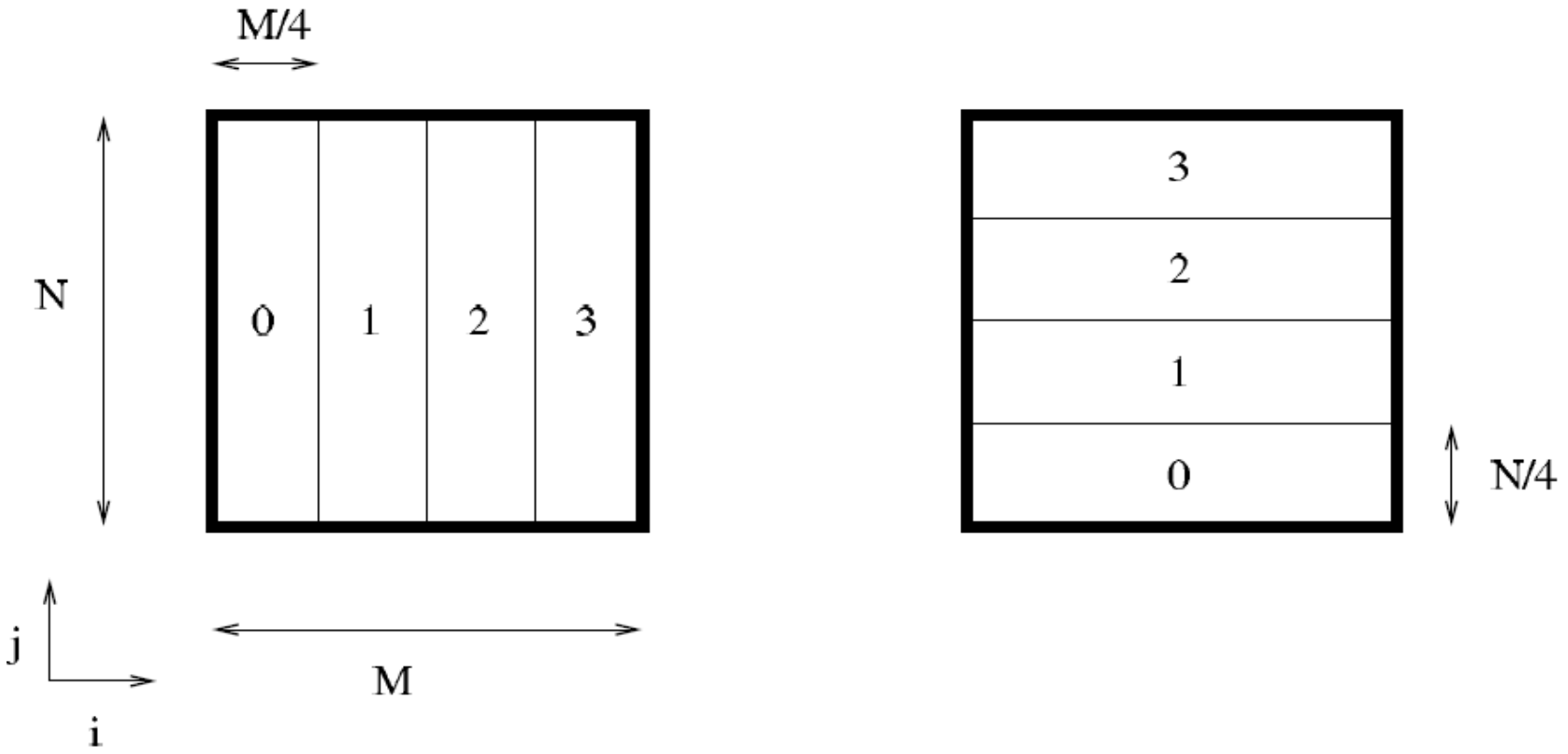
- Uses explicit halos for boundary conditions
  - must use the same boundary conditions used when creating edges
  - declares arrays to have bounds 0, 1, ..., *M+1* and 0, 1, ... *N+1*
  - fills boundaries (all pixels with *i*=0, *i*=*M*+1, *j*=0 or *j*=N+1) with 255
  - reads the input (the edge) into an array *buf* with no halos
  - copies *buf* to the centre of the *edge* array
  - applies the equation many times (1500 iterations)
  - copies *old* back to *buf* excluding the halos
  - writes *buf* to an output file
- Simplifications
  - image dimensions and file names specified at compile time

Parallel algorithm

- Split image into slices between all processes
  - divide across the first index *i* in C, second index *j* in Fortran
  - local array sizes are now *MP* and *NP* (plus halos)
    - C: *MP* = *M*/*P*, *NP* = *N*; Fortran: *MP* = *M*, *NP* = *N*/*P*
- Rank 0 reads into *M* x *N masterbuf* array (with no halos)
  - scatter to local *buf* array
- Implement exactly the same calculation as before except:
  - loop limits are *MP* and *NP* (not *M* and *N*)
  - initialise all halo values to 255
  - **before start** of each iteration, swap boundary values as appropriate
- Gather *buf* array back to *masterbuf*
  - Rank 0 writes to file

Domain Decomposition

- Different choices in C and Fortran



Parallel Case Study solution

- Simplifications
  - specifies number of processes *P* at compile time
    - to make array allocation simpler
  - assumes that the image divides exactly between *P*
    - doesn't even check that this is true!
  - uses MPI_Sendrecv
    - you should use **explicit** non-blocking, i.e. MPI_Isend and/or MPI_Irecv

Coursework

- Differences from Case Study
  - use a 2D decomposition, i.e. try to split into many rectangular regions, not restricted to slices
    - requires more halo swaps than before (and they must be non-blocking!)
    - you cannot do the IO using scatter and gather
  - different boundary conditions
    - fixed values for $i = 0$ and $i = M+1$ (as before), but now a function of i
    - periodic for $j = 0$ and $j = N+1$: $image_{i,N+1} = image_{i,1}$; $image_{i,0} = image_{i,N}$
    - have therefore supplied new edge files, e.g. edgenew192x256.pgm
    - new reference serial codes: imagenew.c, imagenew.f90
  - compute the average pixel value at regular intervals
  - stop when Δ parameter is smaller than some threshold (e.g. 0.1)
    - equals the maximum absolute change in any pixel between *new* and *old*

Questions

- What if the image does not decompose exactly?
  - you might want to deal with this case (see tutorial problems)
  - however, perfectly acceptable to quit and say “can’t do this!”
    - can still run on many different process counts
- What about a prime number of processes?
  - on 3 processes, for example, your code should just use a 1x3 or 3x1 decomposition (i.e. slices as for the case study)
    - 1D decomp is a subset of 2D: your code should run without problems
- Should I get the same answer?
  - Yes! The exercise has been designed so that (rather unusually for a parallel program) I would expect you to get exactly the same answer regardless of how many processes you run on

Hints (i)

- Do no worry about doing elegant IO
  - just implement something simple that works
  - e.g. broadcast the whole image to all processes
    - copy relevant section to your local edge array
- Test section by section
  - run on a single process
    - checks for serious bugs!
  - run for zero iterations
    - checks the IO is correct
  - run with no halo swaps
    - checks the calculation is correct
  - run with only horizontal or vertical halo swaps

Hints (ii)

- You don’t need to run for thousands of iterations
  - can set a maximum count for performance testing
  - or use a larger value for Δ
- You are welcome to use your own (larger) images
  - I will upload instructions
- Bring your problems to the tutorial and drop-in sessions!