

# Image Sharpening Example

---

Running a simple parallel program

 EPSRC

 NERC SCIENCE OF THE ENVIRONMENT

 archer

 CRAY  
THE SUPERCOMPUTER COMPANY

 epcc



# Aims (i)

- To familiarise yourself with running parallel programs
- To run a real parallel code (that does file I/O)
  - on different numbers of cores
  - measure the time taken
  - observe increase in performance (Amdahl's law?)
- Acknowledgements
  - algorithm, diagrams and images taken from:
  - *Hypermedia Image Processing Reference*, Bob Fisher, Simon Perkins, Ashley Walker and Erik Wolfart, Department of Artificial Intelligence, University of Edinburgh (1994)



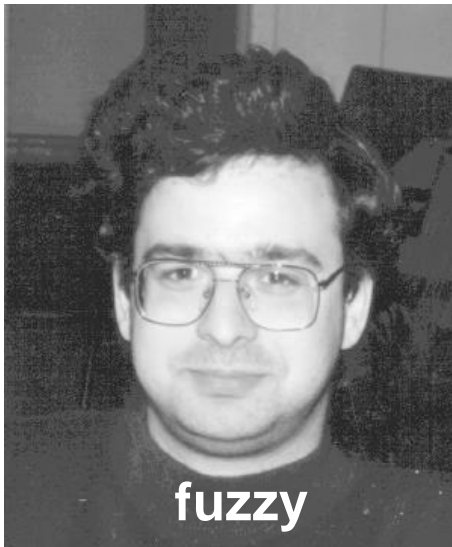
## Aims (ii)

- To get you running on ARCHER
- To sort out all the practical details
  - usernames
  - passwords
  - graphics
  - transferring files
  - compilation
  - using the batch PBS system
  - idiosyncrasies of your Windows / Mac / Linux laptop
  - ...
- Please ask for assistance if you need it!
  - demonstrators are here to help with all aspects of course



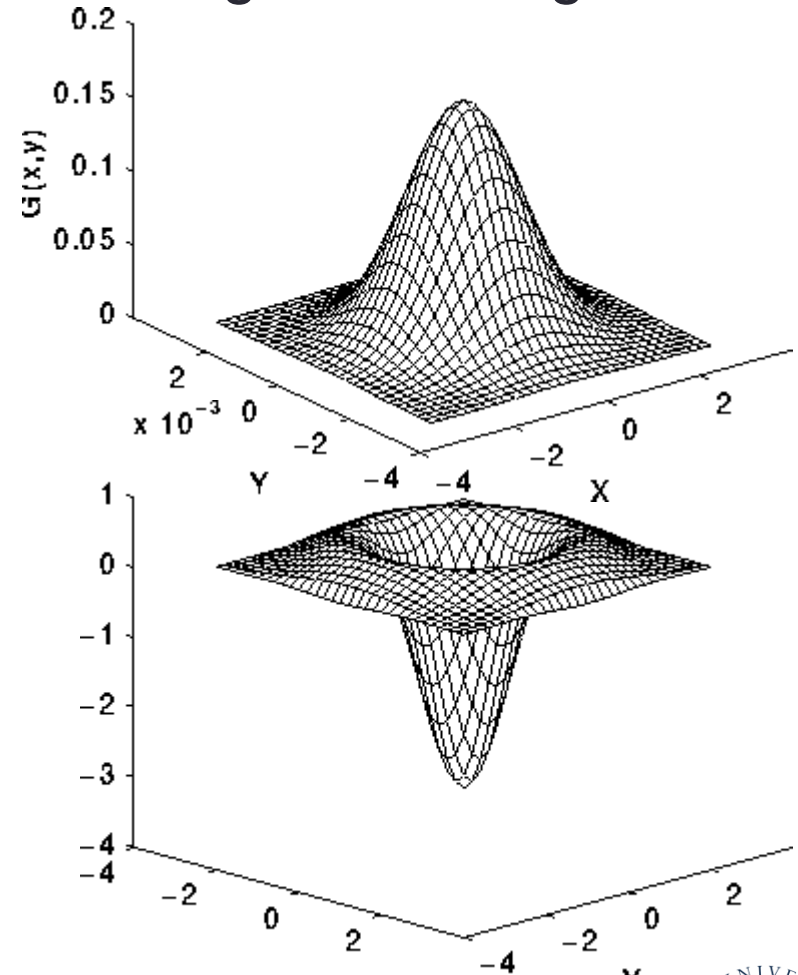
# Image sharpening

- Images can be fuzzy for two main reasons
  - random noise
  - blurring
- Aim to improve quality by
  - smoothing to remove noise
  - detecting edges
  - sharpening up the image with the edges



# Technicalities

- Each pixel replaced by a weighted average of its neighbours
  - weighted by a 2D Gaussian
  - averaged over a square region
- we will use:
  - Gaussian width of 1.4
  - a 17x17 square
  - then apply a Laplacian
    - this detects edges
    - a 2D second-derivative  $\nabla^2$
- Combine both operations
  - produces a single convolution filter



# Implementation

- For over every pixel in the image
  - loop over all pixels in the 17x17 square surrounding it
  - add in the value of the pixel weighted by a filter

$$edge(i, j) = \sum_{k,l=-8,8} image(i+k, j+l) \times filter(k, l)$$

- This gives the edges
  - add the edges back into the original image with some scaling factor
    - we use 2.0
  - rescale the sharpened image so pixels lie in the range 0 - 255

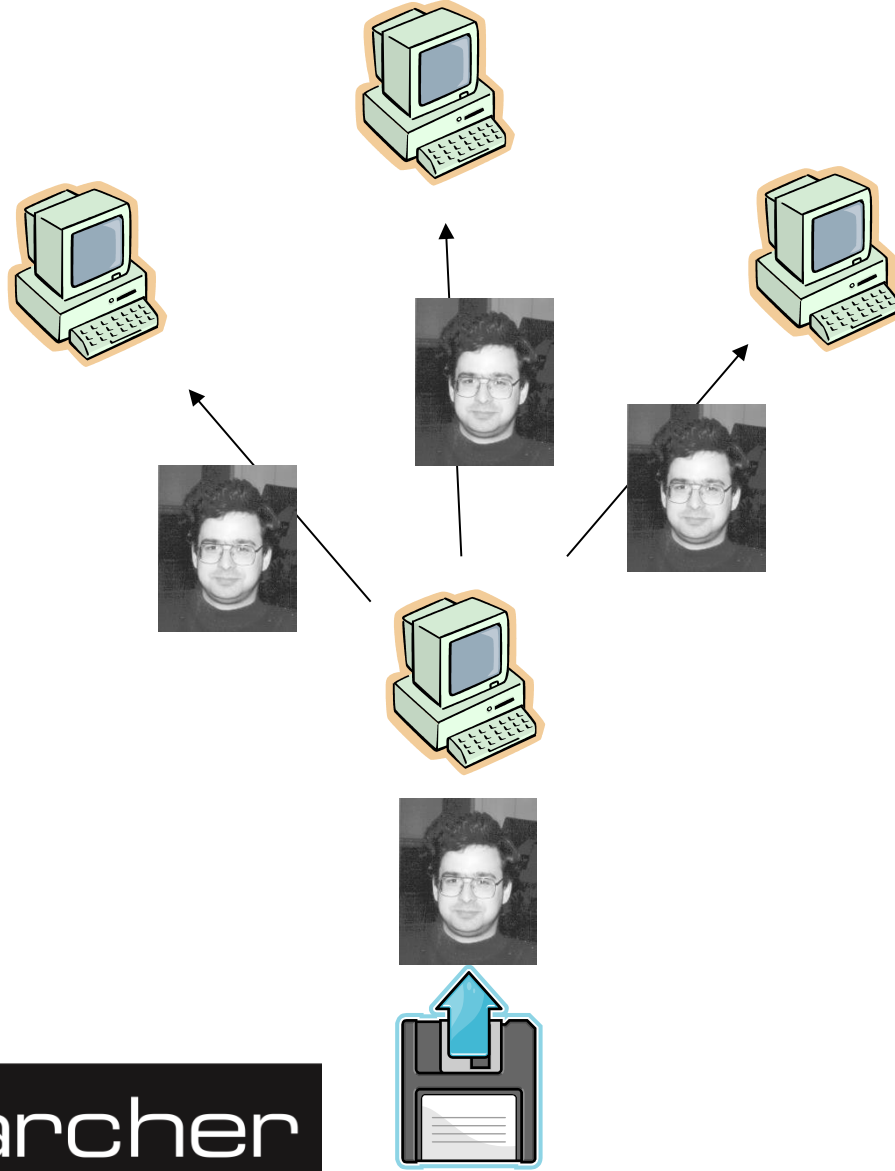


# Parallelisation

- Each pixel can be processed independently
- A master process reads the image
- Broadcast the whole image to every processor
- Each processor computes edges for a subset of pixels:
  - scan the image line by line
  - with four processors, each processor computes every fourth pixel
- Combine the edges back onto a master process
  - add back into original image and rescale
  - save to disk
- Reports two times:
  - calculation time for just computing edges on each processor
  - overall time for the whole program including IO



# Parallelisation



1	2	3	4	1	
2	3	4	1	2	
3					





# Compiling and Running

- We provide a tar file with code and sample images
  - one pair of codes uses MPI and Fortran/C
  - the other pair uses OpenSHMEM and Fortran/C
- You should:
  - copy tar file it to your local account
  - unpack it
  - compile it
  - run it on the back end using appropriate batch scripts
  - view the input and output images using display
  - note the times for different numbers of processors
    - can you interpret them?
- See the exercise sheet for full details!



- Log on to ARCHER and compile and run a code.
- Password: will be provided
- Reservation ID: will be provided
- [http://tinyurl.com/archer170314/sharpen\\_practical.pdf](http://tinyurl.com/archer170314/sharpen_practical.pdf)
- If you are using Windows or do not have SSH installed you will need to obtain an SSH client. One such client is Putty, which can be obtained here (or just search for it on the internet). Also need an X server for graphics, e.g. Xming.
- <http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe>
- <http://sourceforge.net/projects/xming/>

