

Conclusions

"The Angry Penguin", used under creative commons licence from Swantje Hess and Jannis Pohlmann.



Take Aways

- We've seen several approaches to parallel programming
 - They all have advantages and disadvantages
- Threads/OpenMP - data is easily shared between processors, but you have to make sure that two processors don't interfere with each other
- MPI - every processor has it's own copy of data, but you have to arrange for it to get that data somehow
- GPUs - Substantial computational power, but need scope for massive parallelism and have to program and think in a different way

What should I do?

- If you're here to help understand a code that you are working with, look at the technology in that code
- If you need to access data in parallel in a "patchy" fashion (i.e. you don't know what data any particular processor will want) then threads/OpenMP
- If you can split up your problem into discrete chunks that can be worked on nearly independently then MPI will work, but if OpenMP is easy and speeds your problem up enough use that instead

What should I do?

- If you have a problem with opportunities for many parallel operations, and you tend to do the same work on every operation then GPUs might be best
- If you are using a library look to see what options are already available to you, you might not have to do anything!
- The next slide has links to our other courses giving more details on this subject
- There are also many good tutorials on the internet - parallel programming is not uncommon nowadays!

Where should I go?

- <https://warwick.ac.uk/research/rtp/sc/rse/training/intrompi> - Introduction to MPI
- <https://warwick.ac.uk/research/rtp/sc/rse/training/openmp> - Introduction to OpenMP
- <https://github.com/WarwickRSE/ParallelismPrimer> - Examples to go with this course
- Because we currently use NVIDIA provided material, we don't have links to GPU training
 - <https://developer.nvidia.com/blog/even-easier-introduction-cuda/>