HPC at Warwick and Beyond
Section 3 - Essential Snippets

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

Warwick RSE

03/2023

Groundwork

Really Important Note

- Originally we said “HPC” is anything that slows your desktop down for an hour or more
- This includes “I want to run N copies of a single core program”
- BUT a lot of HPC kit is WORSE than your local machine for this
  - E.g. 2.25 GHz (Archer 2) v.s 3-4GHz for a decent-spec’d desktop and up to 5.8GHz for a top end gaming rig
  - Slower cores, but many of them, with fast interconnect
- Sulis is a bit of an exception, being built for "ensemble computing” specifically lots of single core jobs

Processor speeds haven’t been increasing nearly as much in recent years, but you can get a laptop or desktop in the 3.5 GHz range for under £1000.

Essential Terms

- Core-hour
  - Number of cores * number of hours they're needed/used for
  - Usual way of describing resource request
- Walltime
  - Actual clock time a job runs for
- 16 cores for one hour = 16 core hours. Ditto 1 core for 16 hours

Remember there's 8760 hours in a year. Later we talk about applying for compute time and this tends to think in hundred-thousands or millions of core-hours over one year. That means using 115 cores non-stop. Practically, with all queueing, downtime etc, you can rely on your jobs running for upto 50-80% of the walltime in a year, no more. With very careful management of your scheduling you might get 90%. This is NOT the Cluster uptime, which usually target 99% or even five-nines (99.999%). This is a rough estimate of how all the scheduling tends to work out. To use a million core hours in a year, you'd want to be able to use roughly 200 cores at a time, either all in one job, or 4 jobs on 50 cores or however your problem works.

Rules of Thumb

Big and Small

- Big and small jobs
  - Job that could run on specialist workstation is small HPC (8-32 cores)
  - Medium can run easily on local HPC 32-256
  - Large is 256+ cores
- Short and long jobs
  - Short jobs take a few hours - tiny jobs run over a coffee break
  - Medium jobs hours to days
  - Days to weeks is long - unusual to run uninterrupted

No exact answers, but some useful numbers to have in mind
Small, medium, large is always relative. On national facilities, 64 cores might be ‘small’
Despite the growth in the size of national supercomputing facilities the size of typical jobs hasn’t grown much.

Big and Small

- Memory requirements
  - At time of writing, I'd call < 500MB/processor low memory requirement
  - 500MB - 4 or 6 GB is moderate
  - High memory requirements is over 6 or 8GB

What this means is very very problem dependent: these numbers are based on what's available in the average workstation or cluster machine. I.e high-memory is where you might have to use more cores to get enough memory, or use a special sort of node (e.g. the FAT nodes)
Very large amounts of memory on a single machine is hard to get. The highmem nodes on Avon have 1.5TB. You can find some systems with 4TB of RAM but larger than that is very specialised equipment. At this point maybe try to see if you can use inter-node parallelism just to get more RAM even if it doesn't speed up your code at all.

Big and Small

- Data input/output - volume and rates
  - Low values are measured in 100s of MB or 100s of MB per s
  - Moderate data volume is GB to 10s of GB, or IO at a few GB/s
  - Generally 100s of GB to a few TB is a large volume, and the highest data rates on HPC kit will be in the region of 10s of GB/s
  - More than TB of data is getting pretty big and requires dedicated data management strategies, data reduction plan etc

A good spinning hard drive gets order 500MB/s read-write. An SSD bumps that to 3000MB/s. SSDs are around £75/TB at time of writing, HDD £15-20/TB. Cloud solutions start from about £2/TB/month (other than very long term archive storage)
RAID can increase speeds some. Clusters use dedicated file systems, buffers etc to hit their rates, manage millions of files in a single directory, etc. Note that this last one is probably the worst challenge for a filesystem though - avoid many small files wherever possible!!!

Scaling

Scaling in Parallel

- Some things are called ‘embarrassingly parallel’
  - No harder to run on multiple cores than one
  - Moving from n cores to 2n cores is twice as fast
- Some things can’t scale at all
  - No way to divide up work
  - Can’t use more than one core efficiently
- Most things are in between
- Novel techniques mostly finding ways to make workloads more independent

Scaling in Parallel

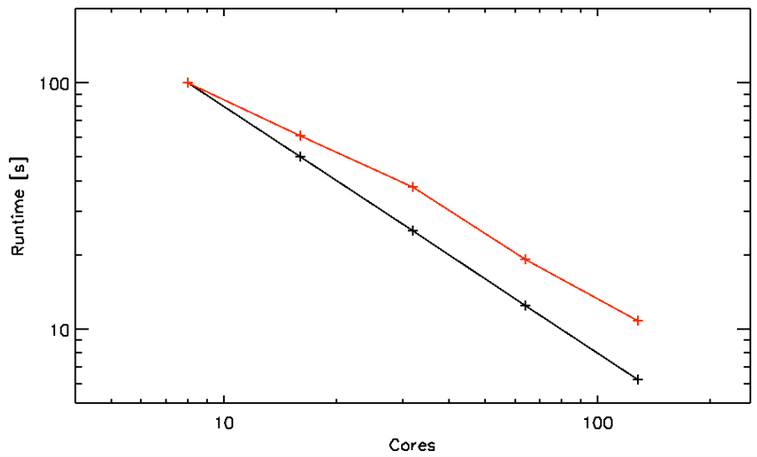
- Speed-up relative to core number is ‘scaling’
- You check scaling to decide how many processors your code can benefit from having
- If code wont scale, there is “no point”* throwing more processors at it
- Nearly always a roll-off in scaling at some point
  - E.g. if splitting a spatial domain, having to share the edges takes time, so there is limit to how small a section is efficient

“If code wont scale, no point throwing more processors at it” see Slide “Amdahl’s law and wasting resources” for more discussion of this

Plotting a Graph

- All sorts of ways to describe scaling
- Some funders etc have a specific requirement
- My favourite is to plot core number against runtime

- Also plot "perfect" scaling
- Line of constant corehours

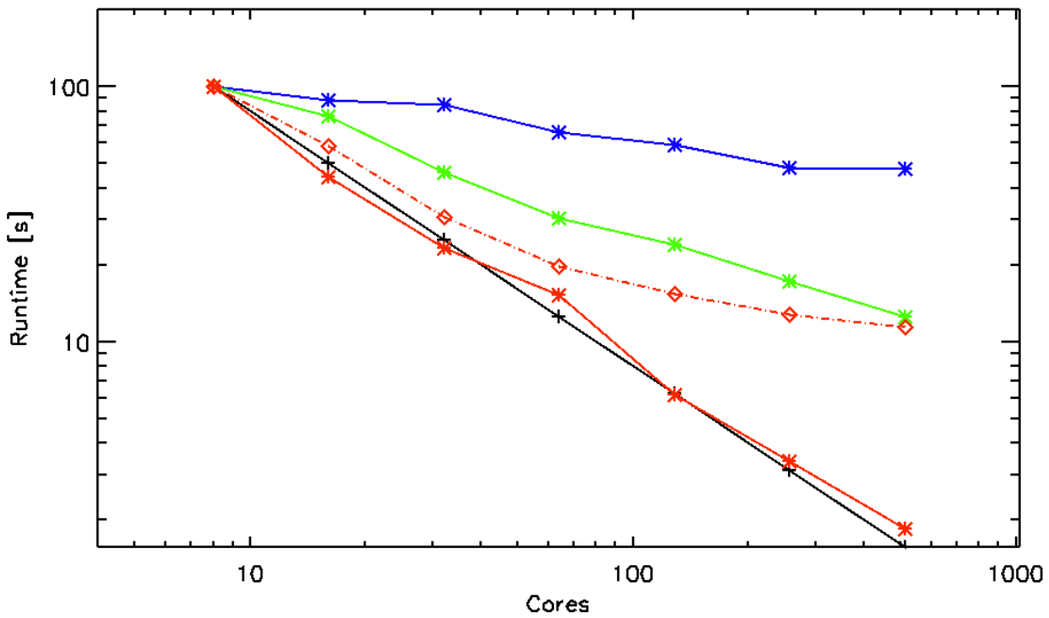


If you're filling in an application, pay attention to what they ask for and do use it. Otherwise, this is a good start.

Note log-log scale. Black line is "perfect". Red is an example line of fair-but-not-perfect scaling, with some random fluctuations.

Plotting a Graph

- Some examples of scaling:



Black is “perfect” scaling. Lying over this (red) is very very good scaling. In fact, sometimes we get slightly over perfect. This is not a problem - everything is relative to out lowest number, so it’s easy to get slight variations like this. Also, sometimes you can get true “superlinear” scaling or even sudden jumps.

The top line is almost flat, i.e. little to no speedup with more cores. The second-top line is also unremarkable at somewhere around half the available speedup (runtime 8x faster, when we use 16x as many cores).

All the lines with crosses show no evidence of “roll-off”. Apart from the noise, they’re straight. The diamonds is a classic roll-off example. (In Amdahl’s law (see below), this one has 90% parallel work. whereas the others just have a sub-optimal response to more resources). Up to 64 cores it looks OK, but after it rapidly tends towards the 10 seconds “best case”

Amdahl's Law

- Formally, imperfect scaling is described by Amdahl's law.
- Part of code is parallel, worked on separately by all processors
- Part is serial, where all processors have to go one by one

E.g. for an MPI code, the core calculation is parallel, but comms and IO are at least partly serial.

Amdahl's Law

- More processors can only affect the time taken by the parallel part
- In fact sometimes the serial part gets worse!
  - Imagine if all processors have to read a specific file
- This means there's a limit to runtime on infinite cores

Note: if you do need all processors to have contents of a file, options include all reading it (either fine, or horrendous depending on core count and file system), one reading and sending to all others, or in some cases, duplicating the file onto all nodes. This operation actually varies from completely parallel to completely serial, depending.

Amdahl's Law

- Imagine having infinite processors
- The time taken for the parallel part of code will be effectively zero!
- Only the serial bit is left
- If say half of code is parallel, can go at best **2x** faster on infinite resources!

Amdahl's Law

- p is fraction of code which is parallel
- s is the relative** number of cores
- S is the resulting speedup
- S = 1/ (1-p + (p/s))

Amdahl's law applies to many sorts of 'resource'. Here we limit to cores and parallelism, but p and s can be other things. In other contexts, s is the speedup of the part which benefits from extra resource

**Note that s is always relative. We're usually interested in Speedup when we double or times ten the number of cores, (s =2, s=10 respectively). Moreover, it is not uncommon for the "effective" number of cores and the "actual" number to not be the same. Even with no "serial" work, it's unusual to get perfect scaling over all core numbers, and not unusual for the effective core number to end up being some complicated function of the actual number.

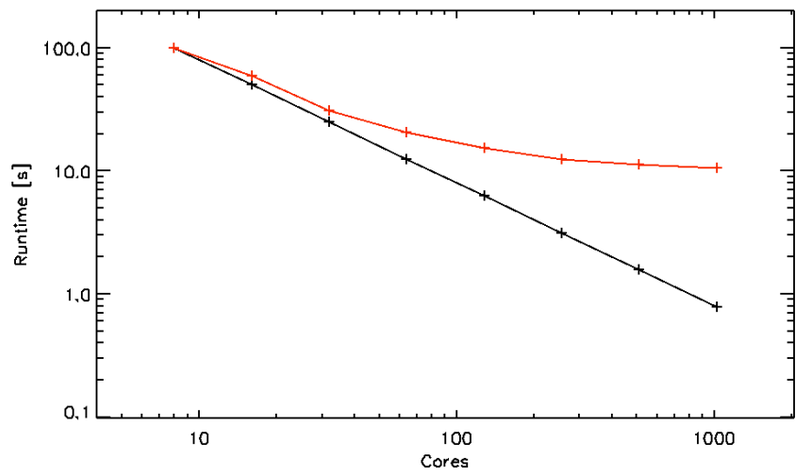
Amdahl's Law and Embarassing Parallelism

- "Embarrassingly parallel" problems have no serial part
- Speedup is S = s
- Double the resources is twice as fast
- This is awesome, but uncommon in true form
- Input/Output usually at least a bit serial

No serial means all parallel, i.e. p = 1. S = 1/ (1/s) = s

Amdahl's Law and Wasting Resources

- As s (resource) goes to infinity, speedup goes to 1/(1-p)
- E.g. half of code parallel p = 0.5 so speedup S→2
- 90% parallel sounds good but...



90% parallel means the best we'll ever get is 10 seconds runtime on infinite cores... That's not great really! In the plot above we also assume 100% scaling of the parallel part, so we roll off quite quickly. In real codes, the situation is both better and worse - we can get **some** benefit from using a larger core number, but we waste even more resources doing so.

Amdahl's Law and Wasting Resources

- It's pretty obviously a waste of resources to keep throwing many cores at a code with limited speedup
  - Roughly 1 - S/s wastage fraction
  - Get the answer faster, but "most" of the core hours "wasted"
  - It's OK to want to get your answer in a reasonable time
  - Balance waiting time against waste of core hours

Sometimes you just want to get your answer ASAP. More commonly your resources are either limited or cost you money, so you have to trade this off. At the same time, your sys-admin is interested in using all available resources 'well'. This means keeping machines as full as possible, but not using cores/memory just for the sake of it.

Worse than Useless

- IMPORTANT - In practice nearly every code will have a roll-off where more cores stops improving even the parallel part
- At this point more cores may or will take ***longer***
- At this point resources are definitely wasted
- Note that some ways of testing scaling can give spuriously low roll-off though

Watch out for this. More than once I have set up a small problem to take a few minutes on 4 cores, done a test run on 128 or so and found awful or even negative (longer on more cores) scaling! The reduced problem was just too small to work well on 128 cores.
E.g. suppose some serial setup. This takes say 1 second per processor. On 4 cores we have 4 seconds and (say) 20 seconds runtime, for 24 seconds total. On 8 cores, we have 8 sec + 20/2, giving 18 seconds. But on 16 cores we have 16 seconds startup and 20/4=5 seconds runtime, for 21 seconds total! 16 cores is **worse** than 8 and 32 cores takes longer than our original 4!!

High Core Count Nodes

- You can now get computers with up to 256 CPU cores per machine
  - We have some with 128 CPU cores
- The memory can't always keep this many cores fed with data, especially if you are doing very little compute per item of data (formally, low **algorithmic intensity**)
- Sometimes you will find that your code can't make efficient use of all the cores even on one machine

Scaling Rules of Thumb

- For a fixed problem, increasing the core number ten-fold, decent starting point is:
  - Excellent scaling is speed up of 9x, i.e. 90% of attainable or better
  - Good is mid 80 % and up
  - Below 70% either there's something wrong, or the problem doesn't really benefit from parallelism

These numbers are completely arbitrary, but a decent starting point. Note that this is for a fixed problem, which is the harder sort of scaling ("strong"). There is an alternative measure where problem size is increased in line with core counts (fixed problem size on each core) ("weak" scaling) which usually gives much higher numbers (say 98, 90, and 80 for the excellent, good and poor thresholds).

Scaling on a New Machine

- Scaling can vary a surprising amount based on machine setups
  - Anything from hardware, memory, storage etc
  - machine settings,
  - versions and builds of libraries
- Try to run a quick scaling test on any new machine if you can

Note: a different desktop with the same specs is not really a “new” machine in this context. On the other hand, when a Cluster gets major software updates it might be.

Profiling

Profiling MPI

- For MPI codes there is a free tool called MpiP
  - Replaces MPI calls with its own monitored ones, so can tell you about calls
- For trickier problems there are commercial tools
  - Arm Forge
  - Intel Advisor

The Arm (formerly Allinea) tools are available on some supercomputers and are very handy. We have it installed on Sulis but not all of our clusters
Intel Advisor tells you things about optimising your code like how well it vectorises. It's available on the clusters, and usually if you have a machine license for the Intel compiler suite.

Error Messages and Help

Error! Error!

- Error messages can be pretty daunting, but there's some useful general rules to get started
- Even if the answer turns out to be trivial, support channels wont mind as long as you've put in a bit of effort
- With experience you develop a "feel" for what can and can't happen

1. Start at the beginning

- Errors multiply and often something small early on leads to a cascade of messages
- Don't worry about later ones until you've fixed the first one
- Can help to redo things “cleanly”. E.g.
  - retype failing command (esp. if copy & paste)
  - new directory
  - make clean & make
  - purge and reload modules

It’s not uncommon to see > 100 errors (most compilers stop counting after some quantity) from a single missing ‘;’ in C++ code.
You might find that you have to divert the messages to a file or pipe through a paging command (https://docs.scrtp.warwick.ac.uk/general-pages/terminal-pages/stioandpipes.html) to even see the first error. It’s always worth doing this, even if you think you know the problem.

2. Remove irrelevant info

- Lots of the message is probably irrelevant
- There's an art to finding the root of an error, but removing some of the fluff always helps
  - BEWARE - if you're wrong about what the problem is, what looks irrelevant might be vital!
- Think about how your internet search would look if your washing machine made a funny noise

E.g. initially you'd probably narrow the problem by looking just for 'funny noise washer'. It might be relevant which part of the cycle the noise happens, or what temperature you used. As you refine the problem, it might start to matter what make and model you have - e.g. there may be a known fault. It's very unlikely that the brand of detergent matters, but what kind might.

3. Work out the general info

- For many sorts of error, it's useful to try and work out the general scope of the problem
  - For example, a bug in code that pops up on a certain number of processors
  - A bug that pops up when file size is over a threshold
  - A bug when you load packages in a specific order only

See previous notes about the washer.

4. Reduce to minimum

- If you're having a problem with writing some code, or with running somebody else's it's very helpful if you can create a Minimum Failing Example
  - The smallest amount of code, input file etc that causes the problem
  - Sometimes you might also have a similar example which works OK - this is great to include
  - Often in doing this you'll find the answer anyway

Still with the washer analogy, if you call out a repairman, you often want to demonstrate the problem, but you'd rather not run a full wash load to do so, or have the repairman unloading your damp underpants.

5. Follow the rules

- Assuming the solution hasn't emerged yet:

1. Work out who might be able to help
   - Is this a cluster problem? A package bug?
2. Read their instructions for posting/messaging
3. Watch for any extra info they request
   - Sometimes there is diagnostic output
4. If you find the answer elsewhere it's polite to share it

Requesting Packages or Programs

Before you Start

- Two ways of getting a package/library/program on most HPC kit
  - Install for you (user) only. E.g. code you just compile
  - Install “globally” for all/many users
- Admins can help with former, but can’t do it for you
- Before requesting something for global use, make sure to consider the following:

Is it actually Global?

- Does this thing **need** to be a global install?
- Would other people use/benefit from it?
- E.g. Python packages are easy to install in user space
  - Doing globally sticks you to the system version
    - Not usually latest bleeding edge version

Is it Open?

- Is it open source, or at least freely released?
- Paid licenses etc may mean it can't be installed as a module
- "Free for education" sometimes doesn't apply to a cluster, and "free for personal use" rarely does
- If not, is there money to buy license(s)?
  - Handling licensing might need paid sys-admin time too

Is there already a good option?

- If this is something you'd like to use, but isn't essential, consider whether there's already a good alternative
  - Or a good user-space alternative
- Everything on clusters has to be supported, kept up to date etc - hard work and takes time
  - More packages = less time for each, updates more irregular

Is it usable?

- Is it actually a good solution, better than the currently available options?
- Is it fairly stable and still being maintained?
- Can you build it on a local machine or does it need a dozen other packages manually installed?
  - If containers are involved, is that feasible/ performant?

Computing Farther Afield

Dedicated Nodes

- If you're applying for grants to do a lot of HPC, you might consider funding a dedicated node
- These are installed in the Taskfarm and reserved for your use
- Use bugzilla to discuss pricing, hosting etc

Midlands Plus and Sulis

- For EPSRC funded research, HPC Midlands plus is next step up (“Tier 2” - national access)
- Warwick hosts the Sulis tier 2 system
  - Can apply for time both as an EPSRC researcher and just as a Warwick researcher
  - Might be able to get more time via EPSRC route

http://www.hpc-midlands-plus.ac.uk/

National Options

- Sulis is one of the EPSRC Tier-2 machines
  - Can also apply for time on the others, either directly or indirectly
    - EPSRC Tier-2 Access calls twice yearly
    - Cirrus http://www.cirrus.ac.uk/access/
    - JADE (machine learning and MD) http://www.jade.ac.uk/access/
    - Materials and Molecular Modelling https://mmmhub.ac.uk/2017/06/14/access/
    - Isambard (Unusual Hardware) https://gw4.ac.uk/isambard/
    - Bede (AI and ML) https://n8cir.org.uk/supporting-research/facilities/nice/

National Options

- STFC has the DiRAC consortium. 1-2 times per year there is a call for applications. Half-a-million core hours typical small project. Up to 100k possible as a seedcorn application, always open
  - https://dirac.ac.uk/callforproposals/
- EPSRC and NERC have allocations on Archer2 via Scientific consortia, and small amounts of time for seedcorn type projects
  - http://www.archer.ac.uk/access/

Very Large Options

- PRACE (partnership for advanced computing in Europe)
  - http://www.prace-ri.eu/prace-project-access/
  - Note MINIMUM requests 15+million core hours over 1 year
    - => 1700 cores continually
- INCITE (US Dept. Energy)
  - “typically cannot be performed anywhere else“
  - 1-5 million discretionary (seed) for porting, etc
  - Average 50+ million. Minimums ~10-20 million
  - Currently only route onto an exascale machine

Note: you almost never want a project to work so that you have to run jobs continually - there’s no time for breaks, downtime, pauses to work out what to run next. And you can’t precisely rely on running any particular number of jobs simultaneously either! If you’re anywhere close to continual running you need a really good formal project management plan in place, detailling exactly what will be run on what date.

Other Options

- BBSRC have some access to Archer2, DiRAC and PRACE
  - See https://bbsrc.ukri.org/research/facilities/#highperformancecomputing
- List of some national and international options is at http://www.hpc-uk.ac.uk/facilities/

Wrap Up

Summary

- Scaling - how code speeds up when given more resources
- Worse scaling means more waste and less scope to trade walltime for cores
- Key to error messages is to work out **what** is happening enough to make it happen deliberately
- Options for HPC exist from 16 core boxes up to 100 million core-hour projects or even larger