Profiling and Analysis Tools

Advanced Parallel Programming

epcc

THE UNIVERSITY OF EDINBURGH

WHAT'S THE PROBLEM?

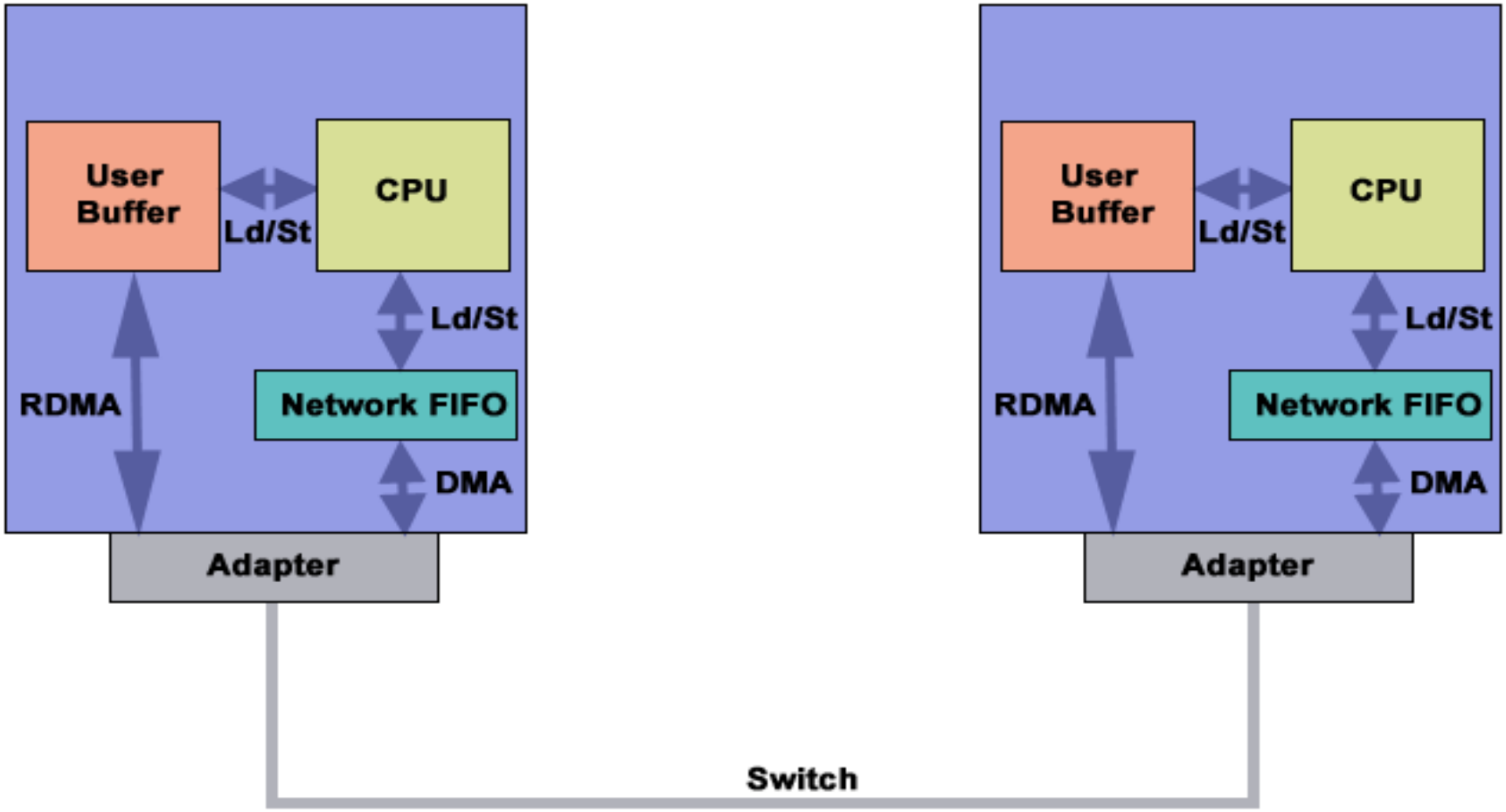
Why do we need tools?

Reminder

Techniques for finding performance problems in a large code:

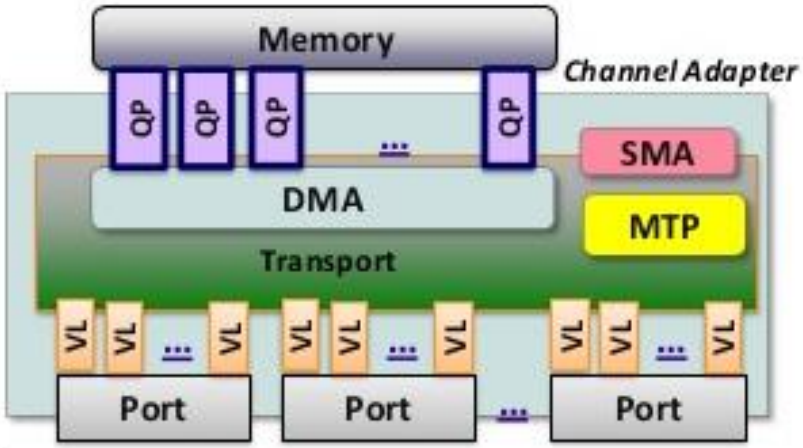
- Manual investigation, looking at the code and machine
- Benchmarking, running and timing the code on a machine
- Profiling tools, sampling and tracing the code on a machine
- Analysis tools, auto-magic wizardry

Simple machine schematic



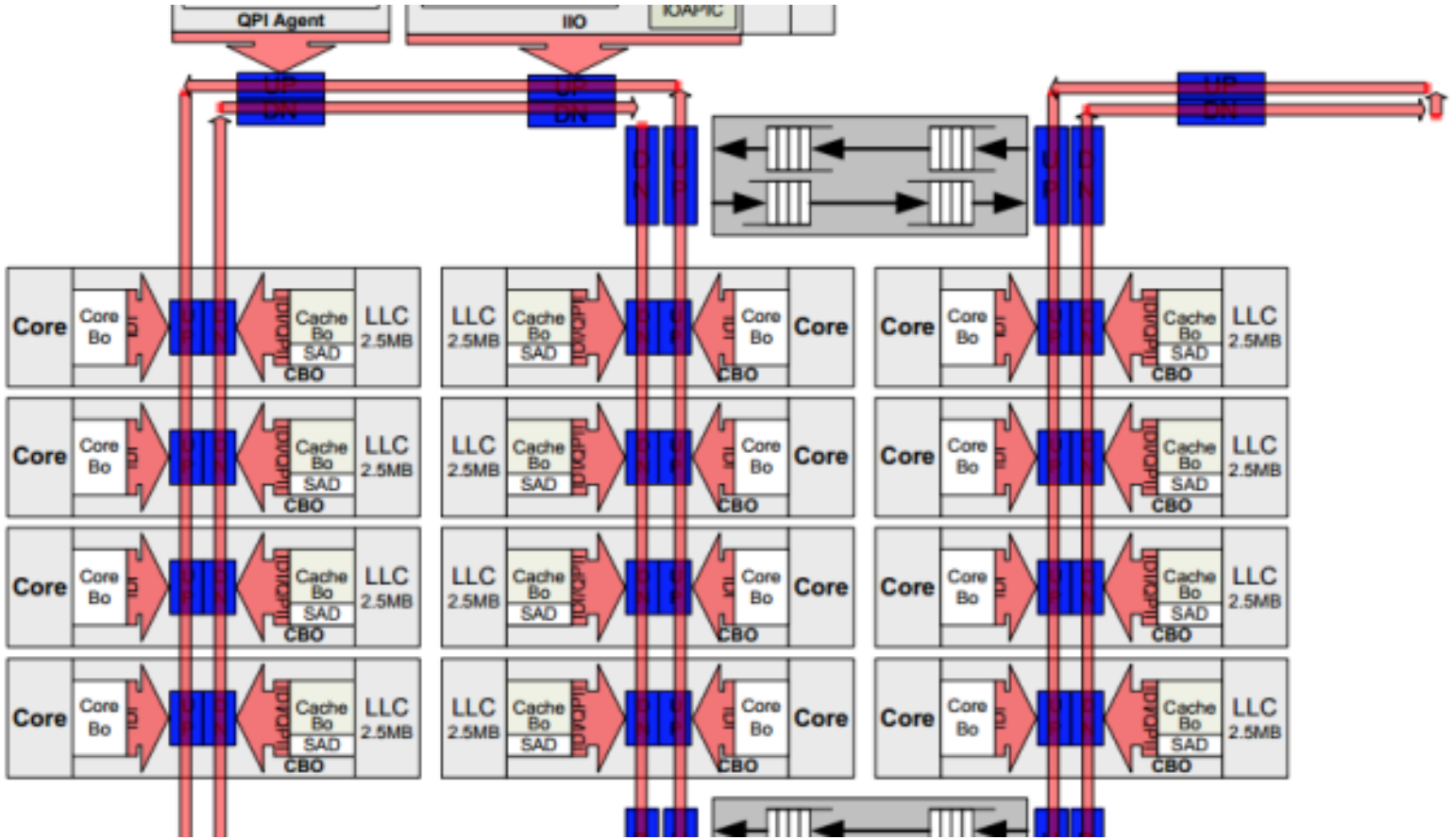
- `https://computing.llnl.gov/tutorials/ibm_sp/`

- Used by processing and I/O units to connect to fabric
- Consume & generate IB packets
- Programmable DMA engines with protection features
- May have multiple ports
  - Independent buffering channeled through Virtual Lanes
- Host Channel Adapters (HCAs)



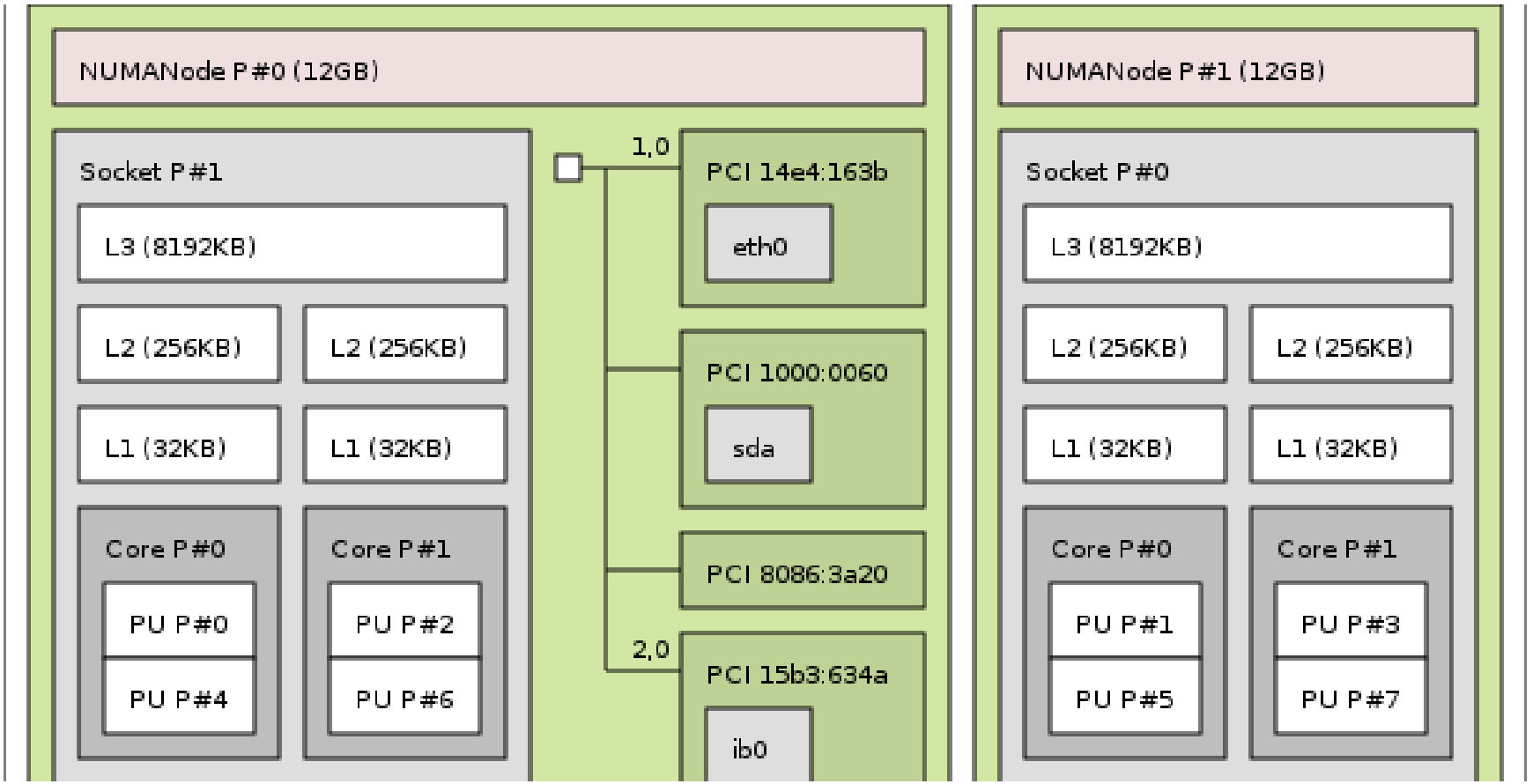
https://image.slidesharecdn.com/ccgrid11ibhselast-160218070646/95/designing-cloud-and-grid-computing-systems-with-infiniband-and-highspeed-ethernet-39-638.jpg

Intel E2607 v3 schematic



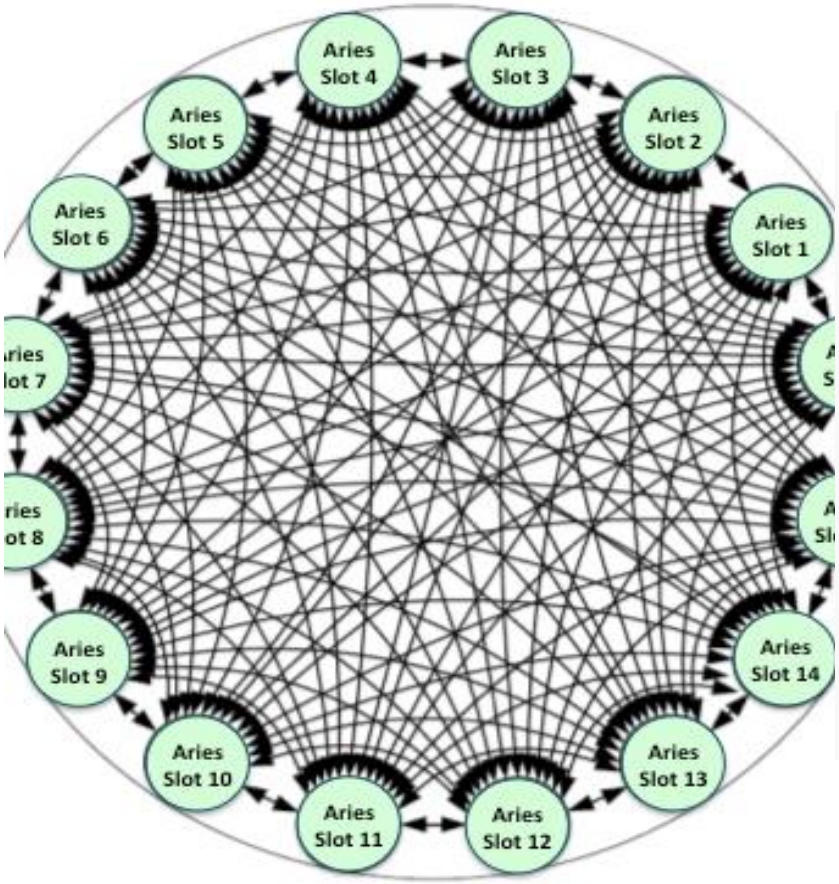
http://www.anandtech.com/show/8584/intel-xeon-e5-2687w-v3-and-e5-2650-v3-review-haswell-ep-with-10-cores

Node hardware

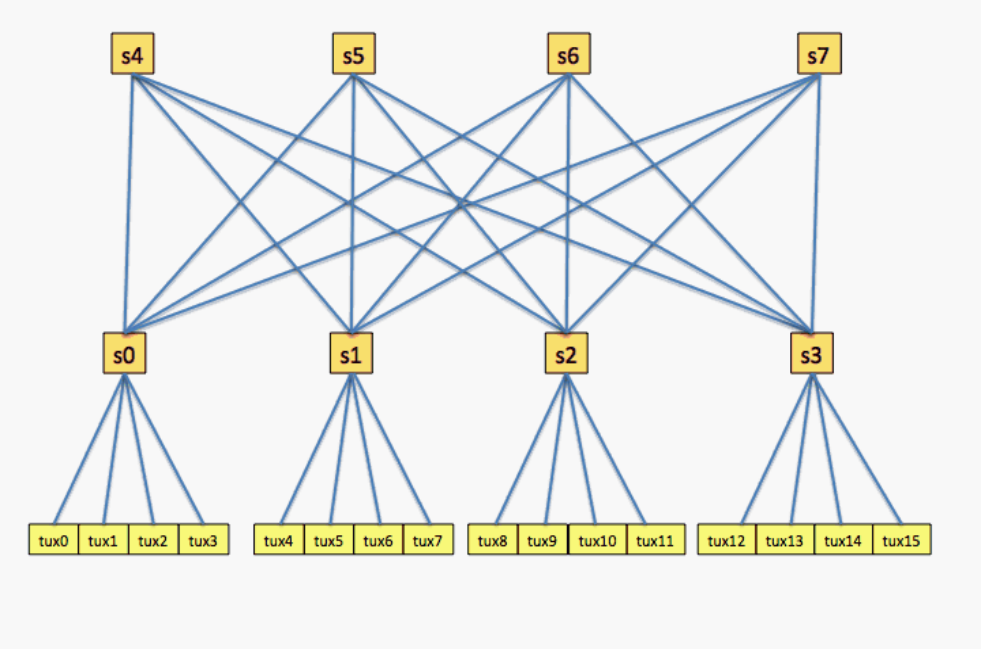


https://www.open-mpi.org/projects/hwloc/

Network tolopogy



Dragonfly topology



Fat tree topology

http://www.nersc.gov/users/computational-systems/edison/configuration/interconnect/

https://slurm.schedmd.com/topology.html

Some useful links

- Information about ARCHER hardware layout:
  - http://www.archer.ac.uk/about-archer/hardware/
- Intel ‘ark’ information for an example processor:
  - http://ark.intel.com/products/75283/Intel-Xeon-Processor-E5-2697-v2-30M-Cache-2_70-GHz
- Information about Cirrus hardware:
  - http://cirrus.readthedocs.io/en/latest/hardware.html
  - https://www.sgi.com/products/servers/ice/ice_xa.html

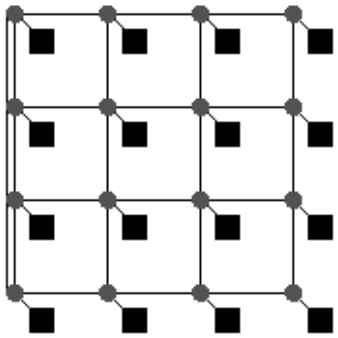
WHY DOES THIS MATTER?

OK, hardware is complicated – so what?

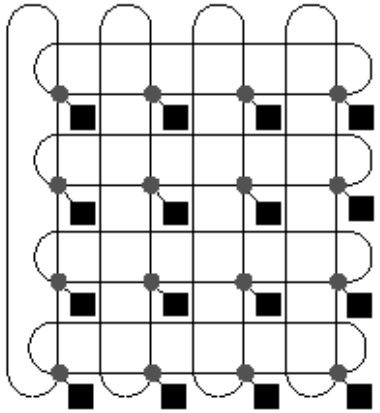
Task mapping

- On most systems, the time taken to send a message between two processors depends on their location on the interconnect.
- Latency depends on number of hops between processors
- Bandwidth might vary between different pairs of processors
- In an SMP cluster, communication is normally faster (lower latency and higher bandwidth) inside a node (using shared memory) than between nodes (using the network)

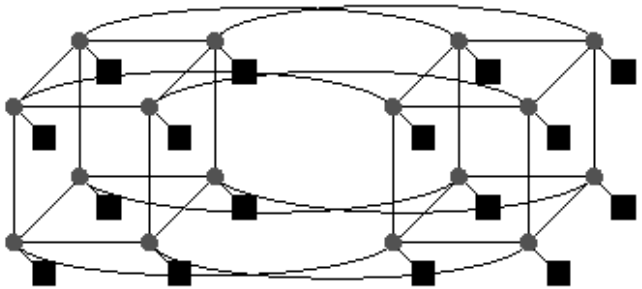
- Communication latency often behaves as a fixed cost + term proportional to number of hops.



a. 2D grid or mesh of 16 nodes



b. 2D torus of 16 nodes



c. Hypercube tree of 16 nodes ($16 = 2^4$ so $n = 4$)

- The mapping of MPI tasks to processors can have an effect on performance
- Want to have tasks which communicate with each other a lot close together in the interconnect.
- No portable mechanism for arranging the mapping.
  - e.g. on Cray XE/XC supply options to aprun
- Can be done (semi-)automatically:
  - run the code and measure how much communication is done between all pairs of tasks
  - tools can help here
  - find a near optimal mapping to minimise communication costs

- On systems with no ability to change the mapping, we can achieve the same effect by create communicators appropriately.
  - assuming we know how `MPI_COMM_WORLD` is mapped
- `MPI_CART_CREATE` has a reorder argument
  - if set to true, allows the implementation to reorder the task to give a sensible mapping for nearest-neighbour communication
  - unfortunately many implementations do nothing, or do strange, non-optimal re-orderings!
- … or use `MPI_COMM_SPLIT`

Custom cluster – no tools

- Basic requirement to ‘pin’ processes/threads
  - Set a “CPU mask” or similar operating system function call
  - Restrict each application thread to a single physical core
- Always possible to schedule one process/thread per core
  - Ensure different runtimes play well together (current research topic)
  - Use as many (or as few) processes as you want
  - Get machine topology by measuring communication performance
  - Chose which processes to use, e.g. based on physical location
- Analysis is mostly guesswork with trial and error
  - Create a small (short time to completion) representative test-case
  - Try to be systematic and cover the available parameter space
  - Keep good records of your tests and the results
- OR install and use tools

WHAT TOOLS ARE THERE?

What can tools do?

Uses for debugging tools

- Where did my program crash?
  - Obtain a stack trace at the point of failure
  - Examine ‘core’ file using gdb (or similar)
  - Use a debugger tool, e.g. Allinea DDT, many others
- Where are the memory leaks in my program?
  - Use ‘valgrind’
- Why does my program get the wrong answer?
  - Use ‘printf’/’write’ statements to verify variable values
  - Use an interactive debug tool to step through code, e.g. DDT/others

Uses for performance tools

- Change process placement to optimise communication
  - Discover and map hardware topology, e.g. hwloc
  - Specify rank mapping, e.g. ‘aprun’ settings or MPI communicators
- Discover ‘hot-spots’ – code that takes up most runtime
  - Identify areas most in need of (greatest impact from) optimisation
  - Profiling tools, trace first, then selectively instrument
  - CrayPAT, Allinea MAP, Scalasca, Intel vTune, TAU, many others
- Discover sub-optimal use of CPU/memory components
  - Access hardware counters, e.g. Performance API (PAPI)
  - Re-order calculation/communication, i.e. algorithm code changes
- Discover sub-optimal communication patterns
  - Infer the problem from other performance evidence, plus intuition
  - Alter calculation/communication, i.e. algorithm code changes

What tools are available?

- Tools on ARCHER:
  - http://www.archer.ac.uk/about-archer/software/
  - “Debugging Tools – DDT, Cray ATP, GDB”
  - “Profiling Tools – CrayPAT”

- Tools on Cirrus:
  - Intel vTune (discovered by doing “module avail”)

- A survey of tools on another machine (Aurora):
  - http://www.paradyn.org/petascale2015/slides/2015_0804_scalableTools_rashawn_knapp_presentation_final.pdf

Current Sample of Tools and Status Overview On Haswell

	Tool/Versions	Description	Status
Low-level tool Foundation	Dyninst 8.2.1	dynamic binary instrumentation tool	GNU and Intel compilations, Test suite completed, Minor change to CMake configuration
	PAPI 5.4.1	interface to sample CPU and off-core performance events	GNU and Intel compilations, Test suite completed, Patch accepted for off-core events
High-level Tools	TAU 2.24.1	profiling and tracing tool for parallel applications, supporting both MPI and OpenMP	Intel Compilation with Intel MPI and Intel C/C++/Fortran compilers, many suite examples tested
	Score-P 1.3	Provides a common interface for high-level tools	2015/16
	Open|Speedshop 2.1	Dynamic Instrumentation tool for Linux: profiling, event tracing for MPI and OpenMP programs. Incorporates Dyninst and PAPI	2015/16
	HPCToolKit 5.3.x r4793	Lightweight sampling measurement tool for HPC; supports PAPI	GNU and Intel compilations with Intel MPI, tests with PAPI and Intel MPI
	Darshan 5.3.2-r4532	IO monitoring tool	2015/16
Low-level Independent	Valgrind 3.10.1	framework for constructing dynamic analysis tools; includes suite of tools including a debugger, and error detection for memory and pthreads.	2015/16
	memcheck	Detects memory errors: stack, heap, memory leaks, and MPI distributed memory. For C and C++.	2015/16
	helgrind	Pthreads error detection: synchronization, incorrect use of pthreads API, potential deadlocks, data races. C, C++, Fortran	2015/16

Summary

- Tools can do *anything* the tool developer can dream up
- There are some well-known tools and many less well-known
- But no standard set of tools that will be available everywhere

- Find out what tools are available on systems you can access
- Read the documentation for each system
- Investigate on the machine itself, e.g. ‘module avail’

- Use tools that are already installed, e.g. by sys admin team
- OR download and install additional tools yourself